Distributed Plan Reconfiguration via Knowledge Transfer in Multi-agent Systems 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 (LTL) 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. While the system runs, each agent updates its knowledge about the workspace via its sensing capability and shares this knowledge with its neighboring agents. Based on this knowledge update, each agent verifies and revises its motion plan in real time. It is ensured that the hard specification is always fulfilled 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.**

I. Introduction

Temporal-logic-based motion planning has gained significant attention in recent years, as it provides a 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 [23]. 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 [1], [5]. Then a high-level discrete plan is found by off-the-shelf model-checking algorithms given the finite transition system and the task specification [3], [11]. This discrete plan is then implemented through the corresponding low-level hybrid controller [20], [22]. The LTL methodology has also been applied for multi-agent systems [10], [18], [30]. 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 [19] and partially-synchronized [20] manner.

Here we instead assume that local task specifications are assigned independently to each agent and there is no pre-specified global task [12], [15]. Since the workspace is only partially-known to each agent initially, the task specification might be infeasible given this initial workspace model and the motion plan might need to be revised whenever the workspace model is updated. Our previous work [15] addresses the first aspect by synthesizing the motion plan that fulfills a potentially infeasible task the most. In this paper, we further improve this technique by considering hard and soft sub-specifications, where the hard specification should be always fulfilled while the soft part can be relaxed. Similar problems are discussed in [27], [28] to find the least-violating strategy under a set of safety rules. However the level of satisfiability is measured different from our approach. In particular, we not only measure how many states along the plan violate the task, but also how much each of those states violates the task. Besides we propose a weight function that balances the task satisfiability of a plan and its implementation cost. Regarding the second aspect, related approaches can be found, e.g., in our earlier work [14] by real-time plan revising, in [24] by local "patching" and in [26] by workspace re-abstraction. In this paper, a new technique is proposed taking into account the fact that the knowledge update comes not only from local sensing functionality but also from inter-agent communications.

The proposed cooperative motion and task planning scheme for multi-agent systems has the following attributes: (1) locally assigned tasks and a partially-known workspace are considered; (2) it is distributed since only local interaction between neighboring agents are assumed; (3) 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; (4) while the hard specification that preserves safety is always guaranteed by the real-time reconfiguration algorithm, the satisfaction of the soft specification is improved gradually by triggering the optimal plan synthesis algorithm in an event-based fashion [16]; (5) it can potentially be applied to many existing partition and motion planning techniques, such as probabilistic roadmap method [10], [17], rapidly-exploring random trees [1], [2], and navigation functions [21].

The rest of the paper is organized as follows: Section II briefly introduces essential preliminaries. In Section III, we describe how to synthesize an initial optimal plan. Section IV focuses on how the knowledge is transferred among the agents. Real-time algorithms to update the system model and the motion plan are provided in Section V. Section VI summarizes the overall structure of the framework and numerical examples are presented in Section VII.

II. Preliminary

Given a team of heterogeneous agents $k \in \mathcal{K} = \{1, 2 \cdots, K\}$, with unique identities (IDs) k, they move within a workspace Π which consists of N regions with unique and pre-defined labels π_n, $n = 1, 2 \cdots, N$.

The authors are with the KTH Centre for Autonomous Systems and ACCESS Linnaeus Center, EES, KTH Royal Institute of Technology, SE-100 44, Stockholm, Sweden. mengg, dimos@kth.se. This work was supported by the Swedish Research Council (VR) and EU STREP RECONFIG: FP7-ICT-2011-9-600825.

A. Local Task Specifications

The basic ingredients of an Linear Temporal Logic (LTL) formula are a set of atomic propositions (APs) AP and several boolean and temporal operators, which are formed according to the following syntax [3]: $\varphi ::= \mathsf{True} \mid a \mid \varphi_1 \wedge \varphi_2 \mid \neg\varphi \mid \mathrm{X}\varphi \mid \varphi_1 \mathrm{U} \varphi_2$, where $a \in AP$ and X (*next*), U (*until*). For brevity, we omit the derivations of other operators like $\Box$ (*always*), $\Diamond$ (*eventually*), $\Rightarrow$ (*implication*) and refer the readers to Chapter 5 of [3]. There is a union of infinite words that satisfy φ: $\mathtt{Words}(\varphi) = \{\sigma \in (2^{AP})^\omega \mid \sigma \models \varphi\}$, where $\models \subseteq (2^{AP})^\omega \times \varphi$ is the satisfaction relation [8].

Furthermore there exists a Nondeterministic Büchi automaton (NBA) $\mathcal{A}_\varphi$ over 2^{AP} corresponding to φ [3], [8]. 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; $\delta : Q \times 2^{AP} \to 2^Q$ is a transition relation and $\mathcal{F} \subseteq Q$ is a set of accepting states. Denote by $\chi(q_m, q_n) = \{l \in 2^{AP} \mid q_n \in \delta(q_m, l)\}$ 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 $\mathtt{Inf}(R) \cap \mathcal{F} \neq \emptyset$, where $\mathtt{Inf}(R)$ is the set of states that appear in R infinitely often [3]. Denote by $\mathcal{L}_\omega(\mathcal{A}_\varphi)$ the accepted language of $\mathcal{A}_\varphi$, which is the set of infinite words that result in an accepting run in $\mathcal{A}_\varphi$. It holds that $\mathtt{Words}(\varphi) = \mathcal{L}_\omega(\mathcal{A}_\varphi)$ [3]. There are fast translation algorithms [13] from an LTL formula to NBA. This process can be done in time and space $2^{\mathcal{O}(|\varphi|)}$.

Denote by $AP_k = \{a_1, a_2, \ldots, a_{|AP_k|}\}$ the set of APs known to agent k. Its 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 APs appearing in φ_k. In particular, we consider the task specification φ_k with the following structure:

$$\varphi_k = \varphi_k^{\text{soft}} \wedge \varphi_k^{\text{hard}}, \tag{1}$$

where φ_k^{soft} and φ_k^{hard} are "soft" and "hard" sub-formulas over AP_k. φ_k^{hard} could include safety constraints like 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 needs for them to be relaxed, while the safety-critical parts should not be relaxed during the process.

B. Agent Description and Partially-known Workspace

The motion of agent k in the workspace is described by a finite-transition system (FTS):

$$\mathcal{T}_k = (\Pi, \longrightarrow_k, \Pi_{0,k}, AP_k, L_k, W_k), \tag{2}$$

where (i) $\longrightarrow_k \subseteq \Pi \times \Pi$ is the transition relation; (ii) $\Pi_{0,k} \subseteq \Pi$ is the set of initial regions where agent k may start; (iii) $L_k : \Pi \to 2^{AP_k}$ is the labelling function that represents agent k's *knowledge*. $L_k(\pi_i)$ is the set of APs satisfied by π_i; (iv) $W_k : \longrightarrow_k \to \mathbb{R}^+$ is the cost of each transition.

We assume that each agent has only partial knowledge of the workspace initially. Thus $\mathcal{T}_k$ might be updated afterwards as described in Section IV-B. Denote by $\mathcal{T}_k^t$ the FTS of agent k at time t. Note that AP_k might be different among the agents due to heterogeneity. For example, in the case study of Section VII, the proposition "`non-fly`" zone can only belong to aerial vehicles and not to ground vehicles, while opposite holds for the proposition "`obstacles`".

We assume that $\mathcal{T}_k$ does not have a terminal state [3]. Then an infinite path of $\mathcal{T}_k$ is an infinite sequence of states $\tau_k = \pi_0\pi_1\pi_2\ldots$ such that $(\pi_i, \pi_{i+1}) \in \longrightarrow_k$ for all $i > 0$. Its trace is defined as the sequence of APs that are true at the states along the path, i.e., $\mathtt{trace}(\tau_k) = L_k(\pi_0)L_k(\pi_1)L_k(\pi_2)\cdots$. Since φ_k^{hard} and φ_k^{soft} are LTL formulas over AP_k, then $\mathtt{trace}(\tau_k) \models \varphi_k^{\text{hard}}$ if and only if $\mathtt{trace}(\tau_k) \in \mathtt{Words}(\varphi_k^{\text{hard}})$ and $\mathtt{trace}(\tau_k) \models \varphi_k^{\text{soft}}$ if and only if $\mathtt{trace}(\tau_k) \in \mathtt{Words}(\varphi_k^{\text{soft}})$.

Definition 1: Given an infinite path $\tau_k = \pi_0\pi_1\pi_2\ldots$ of $\mathcal{T}_k$ and its task φ_k, τ_k is called: (i) **valid** if $(\pi_i, \pi_{i+1}) \in \longrightarrow_k$, for $i = 0, 1, 2\ldots$; (ii) **safe** if $\mathtt{trace}(\tau_k) \models \varphi_k^{\text{hard}}$; (iii) **satisfying** if $\mathtt{trace}(\tau_k) \models \varphi_k$. ■

III. Initial Optimal Plan Synthesis

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. The functions $\chi_1()$ of $\mathcal{A}_k^{\text{hard}}$ and $\chi_2()$ of $\mathcal{A}_k^{\text{soft}}$ are defined analogously as in Section II-A. Now we propose a way to construct the intersection of $\mathcal{A}_k^{\text{hard}}$ and $\mathcal{A}_k^{\text{soft}}$, which is safety-ensured and relaxed.

Definition 2 (Relaxed Automata Composition): The relaxed intersection of $\mathcal{A}_k^{\text{hard}}$ and $\mathcal{A}_k^{\text{soft}}$ is defined by:

$$\tilde{\mathcal{A}}_{\varphi_k} = (Q, 2^{AP_k}, \delta, Q_0, \mathcal{F}), \tag{3}$$

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

Note 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 [3]). An accepting run R of $\tilde{\mathcal{A}}_{\varphi_k}$ intersects with the accepting set $\mathcal{F}$ infinitely often. The last component $t \in \{1, 2\}$ in Q ensures that R has to intersect with both $\mathcal{F}_1 \times Q_2 \times \{1\}$ and $Q_1 \times \mathcal{F}_2 \times \{2\}$. This fact is used in the proof of Theorem 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 $\tilde{\mathcal{A}}_{\varphi_k}$, $R|_{Q_1}$ is an accepting run of $\mathcal{A}_k^{\text{hard}}$. Moreover, $\mathcal{L}_\omega(\tilde{\mathcal{A}}_{\varphi_k}) \subseteq \mathcal{L}_\omega(\mathcal{A}_k^{\text{hard}})$.

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_k 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 $\sigma \in \mathcal{L}_\omega(\tilde{\mathcal{A}}_{\varphi_k})$, σ results in an accepting run of $\tilde{\mathcal{A}}_{\varphi_k}$, 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 $\sigma \in \mathcal{L}_\omega(\tilde{\mathcal{A}}_{\varphi_k})$, $\sigma \in \mathcal{L}_\omega(\mathcal{A}_k^{\text{hard}})$ holds, namely $\mathcal{L}_\omega(\tilde{\mathcal{A}}_{\varphi_k}) \subseteq \mathcal{L}_\omega(\mathcal{A}_k^{\text{hard}})$. ■

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 [10], [11] may fail to provide any solution. Thus we rely on the discrete-plan synthesis algorithm proposed in our earlier work [15] to handle both feasible and potentially infeasible LTL specifications. Firstly, a weighted product Büchi automaton between $\mathcal{T}_k$ and $\tilde{\mathcal{A}}_{\varphi_k}$ needs to be constructed in the following way.

Definition 3 (Weighted Product Automaton): The weighted product Büchi automaton $\tilde{\mathcal{A}}_{p,k} = \mathcal{T}_k \times \tilde{\mathcal{A}}_{\varphi_k} = (Q', \delta', Q_0', \mathcal{F}', W_p)$ is defined as follows:

- $Q' = \Pi \times Q$ and $q' = \langle \pi, q \rangle$, $\forall \pi \in \Pi$ and $\forall q \in Q$.
- $\delta' : Q' \rightarrow 2^{Q'}$. $\langle \pi_j, q_n \rangle \in \delta'(\langle \pi_i, q_m \rangle)$ iff $(\pi_i, \pi_j) \in \longrightarrow_k$ and $q_n \in \delta(q_m, L_k(\pi_i))$.
- $Q_0' = \Pi_{0,k} \times Q_0$ is the set of initial states.
- $\mathcal{F}' = \Pi \times \mathcal{F}$ is the set of accepting states.
- $W_p : \delta' \rightarrow \mathbb{R}^+$ is the weight function, defined as $W_p(\langle \pi_i, q_m \rangle, \langle \pi_j, q_n \rangle) = W_k(\pi_i, \pi_j) + \alpha \cdot \texttt{Dist}(L_k(\pi_i), \chi_2(q_2, \check{q}_2))$, where $q_m = \langle q_1, q_2, t \rangle$ and $q_n = \langle \check{q}_1, \check{q}_2, \check{t} \rangle$; $\alpha \geq 0$ is a design parameter. ■

We denote by $\tilde{\mathcal{A}}_{p,k}^t$ as the product automaton corresponding to $\mathcal{T}_k^t$ at time t. The detailed definition and examples of computing $\texttt{Dist}()$ can be found in Section III.A of [15]. We briefly restate it here. Firstly, an evaluation function over 2^{AP_k} is defined: $\texttt{Eval} : 2^{AP_k} \rightarrow \{0, 1\}^{|AP_k|}$. $\texttt{Eval}(l) = \nu \Longleftrightarrow [\nu_i] = 1$, if $a_i \in l$ and $[\nu_i] = 0$, if $a_i \notin l$, where $l \in 2^{AP_k}$, $\nu \in \{0, 1\}^{|AP_k|}$ and $i = 1, 2 \cdots, |AP_k|$. Then a metric $(2^{AP_k}, \rho)$ is defined as $\rho(l, l') = \|\nu - \nu'\|_1 = \sum_{i=1}^{|AP_k|} |\nu_i - \nu_i'|$, where $\nu = \texttt{Eval}(l)$, $\nu' = \texttt{Eval}(l')$ and $l, l' \in 2^{AP_k}$. $\|\cdot\|_1$ is the ℓ_1 norm [6]. The distance between an element $l \in 2^{AP_k}$ to a set $\chi \subseteq 2^{AP_k} (\chi \neq \emptyset)$ is given by [6]: $\texttt{Dist}(l, \chi) = 0$, if $l \in \chi$ and $\texttt{Dist}(l, \chi) = \min_{l' \in \chi} \rho(l, l')$ otherwise. Note that $\texttt{Dist}(l, \chi)$ is not defined for $\chi = \emptyset$. Remind that $\chi_2(q_2, \check{q}_2) \neq \emptyset$ by the definition of δ in Definition 2. Moreover, it is easy to see that $\texttt{Dist}(L_k(\pi_i), \chi_1(q_1, \check{q}_1)) = 0$, $\forall \langle \pi_j, q_n \rangle \in \delta'(\langle \pi_i, q_m \rangle)$ from the definition of δ'.

The weight of a transition in δ' consists of two parts: $W_k(\pi_i, \pi_j)$ measures the implementation cost of the transition from π_i to π_j and $\texttt{Dist}(L_k(\pi_i), \chi_2(q_2, \check{q}_2))$ 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 R_k is an accepting run of $\tilde{\mathcal{A}}_{p,k}$. Its projection on Π, $\tau_k = R_k|_\Pi$, is both **valid** and **safe** for $\mathcal{T}_k$ and φ_k by Definition 1.

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

Algorithm 1: Optimal Plan Synthesis `OptiPlan( )`

Input: $\mathcal{T}_k$, $\tilde{\mathcal{A}}_{\varphi_k}$
Output: $R_{\text{opt},k}$, $\tau_{\text{opt},k}$

1 Choose the desired α and γ;
2 Compute $\tilde{\mathcal{A}}_{p,k} = \mathcal{T}_k \times \tilde{\mathcal{A}}_{\varphi_k}$ by Definition 3 ;
3 Derive $R_{\text{opt},k}$ in $\tilde{\mathcal{A}}_{p,k}$ by Algorithm 1 in [15]. ;
4 $\tau_{\text{opt},k} = R_{\text{opt},k}|_\Pi$

Furthermore, in order to measure the implementation cost of different accepting runs of $\tilde{\mathcal{A}}_{p,k}$ and how much they violate the soft specification, we consider the accepting runs with the following *prefix-suffix* structure: $R_k = q_0' q_1' \cdots [q_r' q_{r+1}' \cdots \cdots q_n']^\omega$, where $q_0' \in Q_0'$ and $q_r' \in \mathcal{F}'$. The prefix part $(q_0' q_1' \cdots q_{r-1}')$ of R_k from an initial state q_0' to one accepting state q_r' is executed only once. The suffix part $(q_r' q_{r+1}' \cdots \cdots q_n')$ of R_k from q_r' back to itself is repeated infinitely. An accepting run with the prefix-suffix structure has a finite representation and the total cost of an accepting run R_k of $\tilde{\mathcal{A}}_{p,k}$: $\texttt{Cost}(R_k, \tilde{\mathcal{A}}_{p,k}) = \sum_{i=0}^{r-1} W_p(q_i', q_{i+1}') + \gamma \sum_{i=r}^{n-1} W_p(q_i', q_{i+1}') = \texttt{cost}_{\tau_k} + \alpha \cdot \texttt{dist}_{\varphi_k^{\text{soft}}}$, where $\gamma \geq 0$, $\texttt{cost}_{\tau_k} = (\sum_{i=0}^{r-1} + \gamma \sum_{i=r}^{n-1}) W_k(\pi_i, \pi_{i+1})$ is the accumulated implementation cost of the motion plan $\tau_k = R_k|_\Pi$; $\texttt{dist}_{\varphi_k^{\text{soft}}} = (\sum_{i=0}^{r-1} + \gamma \sum_{i=r}^{n-1}) \texttt{Dist}(L_k(\pi_i), \chi_2(q_i'|_{Q_2}, q_{i+1}'|_{Q_2}))$ is the accumulated distance of τ_k with respect to $\mathcal{A}_{\varphi_k^{\text{soft}}}$, $q_i'|_{Q_2}$ and $q_{i+1}'|_{Q_2}$ are the projection of q_i' and q_{i+1}' onto Q_2. 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 [30].

Definition 4 (Optimal Motion Plan): The accepting run of $\tilde{\mathcal{A}}_{p,k}$ that minimizes $\texttt{Cost}(R_k, \tilde{\mathcal{A}}_{p,k})$ is called the optimal accepting run and denoted by $R_{\text{opt},k}$. Its projection onto Π, $\tau_{\text{opt},k} = R_{\text{opt},k}|_\Pi$, is called the optimal motion plan. ■

Given the desired value of α and γ, Algorithm 1 in [15] generates the optimal accepting run and the corresponding optimal plan in the prefix-suffix format. We omit the complete algorithm here due to limited space. Briefly speaking, it utilizes Dijkstra's algorithm [23] for computing the shortest path from any initial state to an accepting state and the the shortest cycle containing this accepting state. The above procedure is summarized in Algorithm 1 as the initial optimal plan synthesis algorithm. By Theorem 2, $\tau_{\text{opt},k}$ is always valid and safe no matter how the value of α and γ are chosen.

Remark 1: Algorithm 1 can be applied directly when φ_k^{soft} is feasible without any modification. Because when α is large enough, i.e., the penalty on violating $\mathcal{A}_{\varphi_k^{\text{soft}}}$ is severe, Algorithm 1 will select the accepting run that satisfies φ_k^{soft}.

Algorithm 2: Transfer Knowledge, `TranKnow( )`

Input: $\mathbf{Sense}_k^t$, $\mathbf{Request}_{g,k}^{t_0}$
Output: $\mathbf{Reply}_{k,g}^t$
forall $\mathbf{Request}_{g,k}^{t_0}$ *received at* $t_0 < t$ **do**
  if $g \in \mathcal{K}_k^t$ **then**
    forall $a \in \varphi_g|_{AP_g}$ **do**
      if $(a, \tilde{\Pi}_a, \tilde{\Pi}_{a,\neg}) \in B(t)$ *of* $\mathbf{Sense}_k^t$ **then**
        add $(a, \tilde{\Pi}_a, \tilde{\Pi}_{a,\neg})$ to $\mathbf{Reply}_{k,g}^t$

IV. Knowledge Update and Transfer

The agents have both the sensing ability to discover the workspace and the communication functionality to share knowledge with their neighboring agents. Denote by $\mathbf{Sense}_k^t = \{A(t), A_\neg(t), B(t)\}$ as the set of sensing information obtained at time $t \geq 0$ by agent k. For simplicity, we assume that any information is gathered when an agent reaches a region, not during the transition from one region to another. It consists of three parts [14]:

- Type-A information: $((\pi_i, \pi_j), w) \in A(t)$ if a new transition (π_i, π_j) is added to $\longrightarrow_k^t$ with weight w, or the weight of an existing transition (π_i, π_j) is updated to w. $(\pi_i, \pi_j) \in A_\neg(t)$ if (π_i, π_j) needs to be removed from $\longrightarrow_k^t$.
- Type-B information: denote $(a, \tilde{\Pi}_a, \tilde{\Pi}_{a,\neg}) \in B(t)$ where $a \in AP_k$; $\tilde{\Pi}_a \subseteq \Pi$ is the set of regions that satisfy proposition a; $\tilde{\Pi}_{a,\neg} \subseteq \Pi$ is the set of regions that do not satisfy proposition a.

This sensing functionality can be modelled by assigning a sensing radius $S_k \geq 0$, such that all regions lying in the sphere $\{\mathbf{p} \in \mathbb{R}^n \,|\, |\mathbf{p} - \mathbf{p}_k^t| \leq S_k\}$ are visible, where $\mathbf{p}_k^t \in \mathbb{R}^n$ is agent k's position at time t.

A. Knowledge Transfer

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

1) Communication Network: The communication network represents how the information flows among the agents. Each agent k has a set of neighboring 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: (1) global communication with a fixed topology; (2) 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., $|\mathbf{p}_g^t - \mathbf{p}_k^t| \leq C_k$ where $\mathbf{p}_g^t$, $\mathbf{p}_k^t \in \mathbb{R}^n$ are the positions of agents g and k at time t. Then $\mathcal{K}_k^t = \{g \in \mathcal{K} \,|\, |\mathbf{p}_g^t - \mathbf{p}_k^t| \leq C_k\}$ is the time-varying neighboring set of agent k at time t.

2) Communication Protocol: Agent k is interested in all 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 $k \in \mathcal{K}_g^t$ for the *first* time at time t, it follows the *subscribing* procedure: agent g sends a request message to agent k that $\mathbf{Request}_{g,k}^t = \varphi_g|_{AP_g}$, which informs agent k the set of propositions agent g is interested. Each agent has a subscriber list, containing the request messages it has received. Note that each agent sends a request to any of its neighboring 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 $\mathbf{Sense}_k^t$, 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, for example regarding proposition a, then agent k checks if $g \in \mathcal{K}_k^t$. If so, it publishes a reply message to agent g that $\mathbf{Reply}_{k,g}^t = \{(a, \tilde{\Pi}_a, \tilde{\Pi}_{a,\neg})\}$, where $a \in \varphi_g|_{AP_g}$; $\tilde{\Pi}_a, \tilde{\Pi}_{a,\neg} \subseteq \Pi$ are defined as in $B(t)$. The above procedure is summarized in Algorithm 2. Note that through this communication mechanism, any request only needs to be sent once and every reply message contains useful knowledge. This subscriber-publisher scheme can be easily implemented by multicast or unicast wireless protocols.

B. Knowledge Update

At time t, agent k might obtain new knowledge from $\mathbf{Sense}_k^t$ and $\mathbf{Reply}_{g,k}^t$, based on which it needs to update its FTS accordingly. Denote by $\mathcal{T}_k^{t^-}$ and $\mathcal{T}_k^{t^+}$ as the FTS before and after the update at time t. $\tilde{\Pi}_k^t \subseteq \Pi$ is used to store the regions of which the labelling function is changed during the update, which serves as an input argument to Algorithm 4 later. Algorithm 3 describes how to construct $\mathcal{T}_k^{t^+}$ from $\mathcal{T}_k^{t^-}$ using $\mathbf{Sense}_k^t$ and $\mathbf{Reply}_{g,k}^t$. In lines 5 and 6, for regions inside $\tilde{\Pi}_a$, it checks if proposition a belongs to $L_k^{t^-}(\pi)$. If not, it adds a to $L_k^{t^-}(\pi)$ and adds π to $\tilde{\Pi}_k^t$. Similar procedure is applied to $\tilde{\Pi}_{a,\neg}$ in lines 7 to 8. Note that if both $\mathbf{Sense}_k^t$ and $\mathbf{Reply}_{g,k}^t$ are empty, $\mathcal{T}_k^{t^+}$ remains the same as $\mathcal{T}_k^{t^-}$.

V. Real-time Reconfiguration

Since $\mathcal{T}_k^t$ might be updated as described in Section IV, the optimal accepting run and its corresponding optimal motion plan derived by Algorithm 1 in Section III need to be evaluated regarding its validity, safety and optimality.

A. Product Automaton Update

Denote by $\tilde{\mathcal{A}}_{p,k}^{t^-}$ and $\tilde{\mathcal{A}}_{p,k}^{t^+}$ as the product automaton before and after the update at time t. It is possible to reconstruct completely $\tilde{\mathcal{A}}_{p,k}^{t^+}$ by Definition 3 using $\tilde{\mathcal{A}}_{\varphi_k}$ and the updated $\mathcal{T}_k^{t^+}$ from Algorithm 3. Here we propose to revise $\tilde{\mathcal{A}}_{p,k}^{t^-}$ locally based on the latest changes in $\mathcal{T}_k^t$. Because the the number of states in $\tilde{\mathcal{A}}_{p,k}$ is *exponential* in the length of φ_k, meaning that it would be computationally expensive to construct $\tilde{\mathcal{A}}_{p,k}$ from scratch each time $\mathcal{T}_k^t$ is updated, in particular given the existence of multi-agent knowledge transfer scheme and the partially-known workspace.

Algorithm 3: Update knowledge, `UpdaKnow( )`

Input: $\mathcal{T}_k^{t^-}$, $\mathbf{Sense}_k^t$, $\mathbf{Reply}_{g,k}^t$.
Output: $\mathcal{T}_k^{t^+}$, $\tilde{\Pi}_k^t$

$\mathbf{Sense}_k^t = \{A(t), A_\neg(t), B(t)\}$;
add (π_i, π_j) to $\longrightarrow_k^{t^-}$, update $W_k((\pi_i, \pi_j))$ to w, $\forall((\pi_i, \pi_j), w) \in A(t)$;
remove (π_i, π_j) from $\longrightarrow_k^{t^-}$, $\forall(\pi_i, \pi_j) \in A_\neg(t)$;
forall $(a, \tilde{\Pi}_a, \tilde{\Pi}_{a,\neg}) \in \mathbf{Reply}_{g,k}^t$ *or* $B(t)$ **do**
  forall $\pi \in \tilde{\Pi}_a$ *and* $a \notin L_k^{t^-}(\pi)$ **do**
    add $\{a\}$ to $L_k^{t^-}(\pi)$, add π to $\tilde{\Pi}_k^t$;
  forall $\pi \in \tilde{\Pi}_{a,\neg}$ *and* $a \in L_k^{t^-}(\pi)$ **do**
    remove $\{a\}$ from $L_k^{t^-}(\pi)$, add π to $\tilde{\Pi}_k^t$;
$\mathcal{T}_k^{t^+} \longleftarrow \mathcal{T}_k^{t^-}$;

Definition 5: After updating $\mathcal{T}_k^{t^-}$ to $\mathcal{T}_k^{t^+}$ by Algorithm 3, a transition $(\langle\pi_i, q_m\rangle, \langle\pi_j, q_n\rangle) \in \delta_k^{'t^-}$ is called: (i) *invalid* if $(\pi_i, \pi_j) \notin \longrightarrow_k^{t^+}$; (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. ■

To update $\tilde{\mathcal{A}}_{p,k}^{t^-}$, all transitions in $\tilde{\mathcal{A}}_{p,k}^{t^-}$ corresponding to removed transitions contained in $A_\neg(t)$ have to be removed. Those are invalid transitions by Definition 5 above. On the other hand, all transitions in $\tilde{\mathcal{A}}_{p,k}^{t^-}$ corresponding to added or modified transitions contained in $A(t)$ and regions with modified labelling function contained in $\tilde{\Pi}_k^t$ need to be reconstructed. The notation Ξ_k^t and $\aleph_k^t$ in Algorithm 4 are used to store the sets of invalid and unsafe transitions, respectively. The above arguments are summarized in Algorithm 4.

Algorithm 4: Update Product Automaton `UpdaProd( )`

Input: $\mathcal{A}_{p,k}^{t^-}$, $\mathcal{T}_k^{t^+}$, $\tilde{\Pi}_k^t$, $\mathbf{Sense}_k^t$.
Output: $\mathcal{A}_{p,k}^{t^+}$, Ξ_k^t, $\aleph_k^t$.

$\mathbf{Sense}_k^t = \{A(t), A_\neg(t), B(t)\}$;
forall $(\pi_i, \pi_j) \in A_\neg(t)$ *and* $q_n \in \delta(q_m, l)$ **do**
  if $\langle\pi_j, q_n\rangle \in \delta_k^{'t^-}(\langle\pi_i, q_m\rangle)$ **then**
    remove $\langle\pi_j, q_n\rangle$ from $\delta_k^{'t^-}(\langle\pi_i, q_m\rangle)$;
    add $(\langle\pi_i, q_m\rangle, \langle\pi_j, q_n\rangle)$ to Ξ_k^t;
forall $q_n \in \delta(q_m, l)$ *and* $((\pi_i, \pi_j), w) \in A(t)$ *or* $(\pi_i, \pi_j) \in \longrightarrow_k^{t^+}$, where $\pi_i \in \tilde{\Pi}_k^t$ **do**
  if $L_k^{t^+}(\pi_i) \in \chi_1(q_m|_{Q_1}, q_n|_{Q_1})$ **then**
    add $\langle\pi_j, q_n\rangle$ to $\delta_k^{'t^-}(\langle\pi_i, q_m\rangle)$;
    recompute $W_p^{t^-}(\langle\pi_i, q_m\rangle, \langle\pi_j, q_n\rangle)$;
  else
    if $\langle\pi_j, q_n\rangle \in \delta_k^{'t^-}(\langle\pi_i, q_m\rangle)$ **then**
      remove $\langle\pi_j, q_n\rangle$ from $\delta_k^{'t^-}(\langle\pi_i, q_m\rangle)$;
      add $(\langle\pi_i, q_m\rangle, \langle\pi_j, q_n\rangle)$ to $\aleph_k^t$;
$\mathcal{A}_{p,k}^{t^+} \longleftarrow \mathcal{A}_{p,k}^{t^-}$;

B. Validity and Safety

Now that we have derived the updated product automaton $\mathcal{A}_{p,k}^{t^+}$, an accepting run $R_k^{t^-}$ of $\tilde{\mathcal{A}}_{p,k}^{t^-}$ and its 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 $\tilde{\mathcal{T}}_k^{t^+}$ and φ_k? Denote by $\mathtt{edge}(R_k^{t^-})$ the set of transitions appearing in $R_k^{t^-}$. The first question is answered by the following theorem.

Theorem 3: Assume $R_k^{t^-}$ is an accepting run of $\tilde{\mathcal{A}}_{p,k}^{t^-}$. Let $\tilde{\mathcal{A}}_{p,k}^{t^-}$ be updated to $\tilde{\mathcal{A}}_{p,k}^{t^+}$ by Algorithm 4. Then (i) $\tau_k^{t^-}$ remains **valid** if and only if $\Xi_k^t \cap \mathtt{edge}(R_k^{t^-}) = \emptyset$; (ii) $\tau_k^{t^-}$ remains **safe** if $\aleph_k^t \cap \mathtt{edge}(R_k^{t^-}) = \emptyset$.

Proof: Since $R_k^{t^-}$ is an accepting run of $\tilde{\mathcal{A}}_{p,k}^{t^-}$, $\tau_k^{t^-}$ is both valid and safe for $\mathcal{T}_k^{t^-}$ by Theorem 2. If $\Xi_k^t \cap \mathtt{edge}(R_k^{t^-}) = \emptyset$ and $\aleph_k^t \cap \mathtt{edge}(R_k^{t^-}) = \emptyset$, $R_k^{t^-}$ does not contain any invalid or unsafe transitions. Thus $R_k^{t^-}$ is still an accepting run of $\tilde{\mathcal{A}}_{p,k}^{t^+}$. Then (i) by Theorem 2, $\tau_k^{t^-}$ is still valid. On the other hand, if $\Xi_k^t \cap \mathtt{edge}(R_k^{t^-}) \neq \emptyset$, $R_k^{t^-}|_\Pi$ contains at least one invalid transition, thus not valid by Definition 1; (ii) by Theorem 2, $\tau_k^{t^-}$ is also safe. ■

One possible solution for the second question could be to recall Algorithm 1 with respect to $\tilde{\mathcal{A}}_{p,k}^{t^+}$, which generates a new optimal accepting run of $\tilde{\mathcal{A}}_{p,k}^{t^+}$. But this method requires a complete new search of $\tilde{\mathcal{A}}_{p,k}^{t^+}$, which is computationally inefficient (the worst-case time complexity being $\mathcal{O}(|\tilde{\mathcal{A}}_{p,k}| \cdot \log|\tilde{\mathcal{A}}_{p,k}| \cdot |Q_0'| \cdot |\mathcal{F}'|)$) in case of large $\mathcal{T}_k$ and complex φ_k, especially when $\tilde{\mathcal{A}}_{p,k}$ has to be updated frequently.

Instead we are interested in revising $R_k^{t^-}$ locally such that it fulfils the accepting condition of $\tilde{\mathcal{A}}_{p,k}^{t^+}$, but not necessarily optimal. The idea is that since most of the path segments belonging to $\tilde{\mathcal{A}}_{p,k}^{t^-}$ remain unchanged for $\tilde{\mathcal{A}}_{p,k}^{t^+}$, we only need to make up the edges that are invalid or unsafe in $\tilde{\mathcal{A}}_{p,k}^{t^+}$. In our earlier work [14], we proposed a real-time revising algorithm to revise an accepting run whenever this accepting run contains invalid transitions due to updates in the product automaton. Function `Revise()` from Algorithm 5 in [14] essentially determines the location of the removed transition in $R_k^{t^-}$ and locally finds a "bridging" segment that make up the removed transition by depth-first search, such that the accepting run becomes valid. The updated $R_k^{t^+}$ is projected onto Π as the motion plan at time t. It is proved by Theorem 3 in [14] that an accepting run of $\tilde{\mathcal{A}}_{p,k}^{t^+}$ can always be found by this algorithm if there exists one. It is summarized in Algorithm 5, where `Revise()` is the main tool.

C. Optimality

Algorithm 5 offers a way to locally revise the invalid or unsafe plan, which however does not maintain the cost optimality as Algorithm 1. 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) in [7]. As also pointed out in both papers, it is inefficient to re-compute the shortest path "from

Algorithm 5: Revise the plan, `ReviPlan( )`

Input: $R_k^{t^-}$, Ξ_k^t, $\aleph_k^t$, $\tilde{\mathcal{A}}_{p,k}^{t^+}$
Output: $R_k^{t^+}$, $\tau_k^{t^+}$

$E^t = \Xi_k^t \cup \aleph_k^t$;
$R_k^{t^+} = \texttt{Revise}(\tilde{\mathcal{A}}_{p,k}^{t^+}, R_k^{t^-}, E^t)$. $\tau_k^{t^+} = R_k^{t^+}|_\Pi$;
`// Algorithm 5 in [14]`

scratch" using the well-known static solution like Dijkstra each time a topology change occurs in the graph. Moreover, the DSPP algorithms proposed in [7] require much more complicated mechanism and sometimes not more efficient than calculating everything from scratch, especially when multiple changes occur simultaneously.

Thus in this paper, we propose the following criterion to ensure the optimality check. Denote by Υ_k^t the accumulated number of number of changes in $\tilde{\mathcal{A}}_{p,k}$: $\Upsilon_k^{t^+} = \Upsilon_k^{t^-} + |\Xi_k^t| + |\aleph_k^t|$. Denote by Δ_k^t the timespan since the last time the optimal planner Algorithm 1 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 condition holds: (1) $\Upsilon_k^t \geq N_k^{\text{call}}$; (2) $\Delta_k^t \geq T_k^{\text{call}}$, Algorithm 1 is called with respect the latest product automaton $\tilde{\mathcal{A}}_{p,k}^t$ to derive the optimal motion plan $\tau_{\text{opt},k}^t$, but using agent k's current position as the initial state in its FTS. Both $\Upsilon_k^{t^+}$ and $\Delta_k^{t^+}$ are reset to zero after Algorithm 1 is called.

VI. Overall Architecture

The overall architecture is summarized in Algorithm 6. When the system starts, each agent synthesizes its own initial motion plan using Algorithm 1. It sends requests to neighboring 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 FTS 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 6 is always valid and safe. Moreover, for any $t' \geq 0$, there exists time $t \in [t', t' + T_k^{\text{call}}]$ such that τ_k^t is the optimal motion plan for $\mathcal{T}_k^t$ and φ_k.

Proof: Whenever Algorithm 1 is called with respect to the latest $\tilde{\mathcal{A}}_{p,k}^{t^+}$, it generates an optimal accepting run $R_{\text{opt},k}^{t^+}$. When Algorithm 1 is not called, Algorithm 5 revises the potentially falsified $R_k^{t^-}$ such that $R_k^{t^+}$ is an accepting run for $\tilde{\mathcal{A}}_{p,k}^{t^+}$. In both cases, it is guaranteed that an accepting run for $\tilde{\mathcal{A}}_{p,k}^{t^+}$ will be found if there exists one. As a result, R_k^t is always an accepting run for $\tilde{\mathcal{A}}_{p,k}^t$, meaning that the corresponding motion plan τ_k^t is always valid and safe. τ_k^t is the optimal motion plan for $\mathcal{T}_k^t$ and φ_k whenever Algorithm 1 is called. Due to the triggering condition in line 11 of Algorithm 6, Algorithm 1 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 motion plan consists of two main aspects: region-to-region navigation and collision avoidance among the agents. As pointed out earlier, it could potentially be combined with many existing partition and motion planning techniques, like vector-field-based [9], navigation function based [12], [21] for linear [4] or nonholonomic dynamic models [25]. The correctness of the proposed solutions follows from Theorem 4 and the correctness of Dijkstra's shortest path algorithm. Let $|\tilde{\mathcal{A}}_{p,k}|$ be the size of $\tilde{\mathcal{A}}_{p,k}$ from Definition 3. Algorithm 1 runs in $\mathcal{O}(|\tilde{\mathcal{A}}_{p,k}| \cdot \log|\tilde{\mathcal{A}}_{p,k}| \cdot |Q_0'| \cdot |\mathcal{F}'|)$. Algorithms 3 and 4 has the complexity linear to the size of $\textbf{Sense}_k^t$. Algorithm 5 has the complexity linear to the length of $R_k^{t^-}$.

Algorithm 6: 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, τ_k^t, Υ_k^t, Δ_k^t

1 **if** $t = 0$ **then**
2 compute $\tilde{\mathcal{A}}_{p,k}^0$ by Definition 3;
3 $(R_{\text{opt},k}^0, \tau_{\text{opt},k}^0) = \texttt{OptiPlan}(\mathcal{T}_k^0, \tilde{\mathcal{A}}_{p,k}^0)$; `// Algorithm 1`
4 send $\textbf{Request}_{k,g}^t$;
5 check $\textbf{Reply}_{h,k}^t$ and $\textbf{Request}_{g,k}^{t_0}$;
6 read $\textbf{Sense}_k^t$ from sensors;
7 send $\textbf{Reply}_{k,g}^t = \texttt{TranKnow}(\textbf{Sense}_k^t, \textbf{Request}_{g,k}^{t_0})$; `// Algorithm 2`
8 $[\mathcal{T}_k^{t^+}, \tilde{\Pi}_k^t] = \texttt{UpdaKnow}(\mathcal{T}_k^t, \textbf{Sense}_k^t, \textbf{Reply}_{g,k}^t)$; `// Algorithm 3`
9 $[\tilde{\mathcal{A}}_{p,k}^{t^+}, \Xi_k^t, \aleph_k^t] = \texttt{UpdaProd}(\tilde{\mathcal{A}}_{p,k}^t, \mathcal{T}_k^{t^+}, \tilde{\Pi}_k^t, \textbf{Sense}_k^t)$; `// Algorithm 4`
10 $\Upsilon_k^{t^+} = \Upsilon_k^t + |\Xi_k^t| + |\aleph_k^t|$;
11 **if** *($\Upsilon_k \geq N_k^{\text{call}}$) or ($\Delta_k^t \geq T_k^{\text{call}}$)* **then**
12 $[R_k^{t^+}, \tau_k^{t^+}] = \texttt{OptiPlan}(\mathcal{T}_k^{t^+}, \tilde{\mathcal{A}}_{p,k}^{t^+})$; `// Algorithm 1`
13 $\Upsilon_k^{t^+} = 0$, $\Delta_k^t = 0$;
14 **else**
15 $[R_k^{t^+}, \tau_k^{t^+}] = \texttt{ReviPlan}(R_k^t, \Xi_k^t, \aleph_k^t, \tilde{\mathcal{A}}_{p,k}^{t^+})$; `// Algorithm 5`
16 $R_k^t = R_k^{t^+}$, $\tau_k^t = \tau_k^{t^+}$, $\Upsilon_k^t = \Upsilon_k^{t^+}$, $\tilde{\mathcal{A}}_{p,k}^t = \tilde{\mathcal{A}}_{p,k}^{t^+}$, $\mathcal{T}_k^t = \mathcal{T}_k^{t^+}$;

VII. Case Study

In the following case study, we apply the framework to a team of 9 autonomous robots: three of them are aerial vehicles that repetitively surveil over base stations; the rest are ground vehicles that collect food and water 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 8GB of RAM).

A. Workspace and Agent Description

As shown Fig. 1, the workspace we consider is a 9×9 square consisting of 5 base stations of size 1×1 (in yellow,

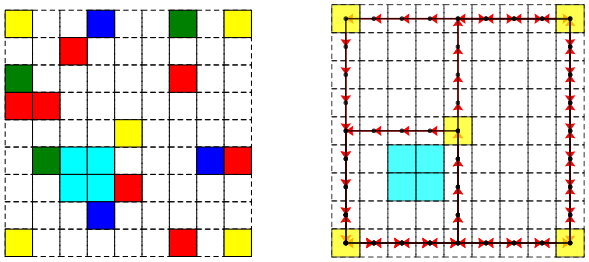

Fig. 1. Left: the actual workspace model as described in Section VII-A. Right: the final optimal motion plan for aerial vehicles, which corresponds to the final optimal plan in Fig. 3. It satisfies both $\varphi^{\text{hard}}_{\text{Ar\_i}}$ and $\varphi^{\text{soft}}_{\text{Ar\_i}}$.

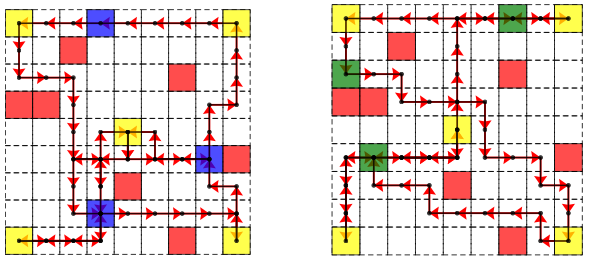

Fig. 2. The final optimal motion plan for agents Gw_i (left) and for agents Gf_i (right). It can be seen that water (in blue) or food (in green) is fetched before any base station and all base stations are visited infinitely often, while at the same time obstacle avoidance is ensured.

denoted by $\texttt{b1}\ldots,\texttt{b5}$): four in the corners and one in the middle. Besides, they are no-fly zone of size 2×2 (in cyan, denoted by `nfly`) and numerous obstacles of size 1×1 (in red, denoted by `obs`). Food (in green, denoted by `food`) and water (in blue, denoted by `water`) resources of size 1×1 are scattered in the free space.

Three aerial vehicles (Ar_1, Ar_2, Ar_3) start randomly from one of the base stations. They have the hard specification "repetitively visit at least one of the base stations, while avoiding the no-fly zone", and soft specification "visit all base stations infinitely often". For aerial vehicles, $\texttt{b1}\ldots,\texttt{b5}$, `nfly`, `water`, `food` are their known propositions. Initially, they do not know the location of base station `b5`, no-fly zone, nor the location of water and food resources. They have a sensing radius of 5 in the x-y coordinates. In LTL formulas, the specifications are given as $\varphi^{\text{hard}}_{\text{Ar\_i}} = (\square\neg\,\texttt{nfly}) \wedge (\square\lozenge(\texttt{b1} \vee \texttt{b2} \vee \texttt{b3} \vee \texttt{b4} \vee \texttt{b5}))$ and $\varphi^{\text{soft}}_{\text{Ar\_i}} = (\square(\lozenge\texttt{b1} \wedge \lozenge\texttt{b2} \wedge \lozenge\texttt{b3} \wedge \lozenge\texttt{b4} \wedge \lozenge\texttt{b5}))$, for $i = 1, 2, 3$. The NBA associated with $\varphi^{\text{hard}}_{\text{Ar\_i}}$ and $\varphi^{\text{soft}}_{\text{Ar\_i}}$ have 2 and 6 states by [13]. The other six robots are ground vehicles: three of them (denoted by Gf_1, Gf_2, Gf_3) collect food and the rest (denoted by Gw_1, Gw_2, Gw_3) for water, to supply the base stations. 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". For ground vehicles, $\texttt{b1}\ldots,\texttt{b5}$, `obs`, `water`, `food` are known propositions. Initially, they start randomly from one base station and only knows the location of one water (or food) resource, but not the obstacles and other base stations. In LTL formulas, the hard and soft specifications for Gw_i are given by $\varphi^{\text{hard}}_{\text{Gw\_i}} = (\square\lozenge\neg\texttt{obs}) \wedge \texttt{Order}$ and $\varphi^{\text{soft}}_{\text{Gw\_i}} = \varphi^{\text{soft}}_{\text{Ar\_i}}$,

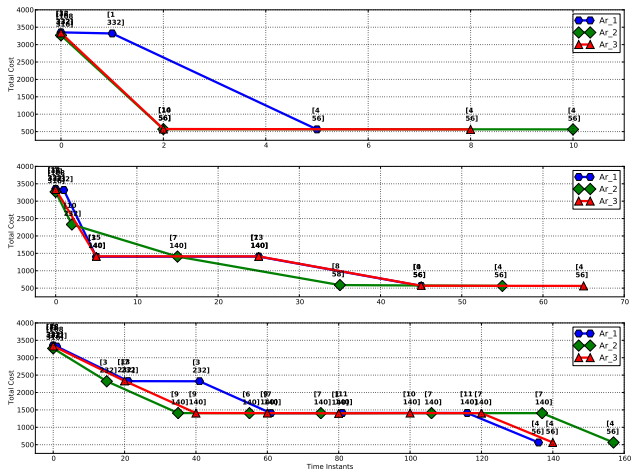

Fig. 3. This figure shows how the total cost of agents' (Ar_1, Ar_2, Ar_3 optimal plan evolves with time under the complete, clustered and dynamic communication topologies (up to down). Each optimal plan is labelled by the cost of its prefix and suffix. The initial plan has a total cost 3352, with prefix and suffix costs being $[32, 332]$. The final plan has a total cost 563, with prefix and suffix costs being $[4, 56]$. Observe the time it takes before they converge to the same final plan.

where $i = 1, 2, 3$; $\texttt{Order} = (\square\lozenge\,\texttt{water}) \wedge (\square(\texttt{water} \Rightarrow \mathbf{X}(\neg\texttt{water}\mathbf{U}(\texttt{b1} \vee \texttt{b2} \vee \texttt{b3} \vee \texttt{b4} \vee \texttt{b5}))) \wedge (\square((\texttt{b1} \vee \texttt{b2} \vee \texttt{b3} \vee \texttt{b4} \vee \texttt{b5}) \Rightarrow \mathbf{X}(\neg(\texttt{b1} \vee \texttt{b2} \vee \texttt{b3} \vee \texttt{b4} \vee \texttt{b5})\mathbf{U}\texttt{water}))$, which says that water must be fetched and supplied to at least one base station infinitely often. $\varphi^{\text{soft}}_{\text{Gw\_i}}$ is the same as for Ar_i, requiring that all base stations be supplied infinitely often. They have a sensing radius of 3 in the x-y coordinates. The NBA associated with $\varphi^{\text{hard}}_{\text{Gw\_i}}$ and $\varphi^{\text{soft}}_{\text{Gw\_i}}$ have 10 and 6 states by [13]. The soft and hard specifications $\varphi^{\text{hard}}_{\text{Gw\_i}}$ and $\varphi^{\text{soft}}_{\text{Gw\_i}}$ for Gw_i can be defined in a similar way by replacing proposition `water` with `food`.

Clearly, the soft specification is impossible to fulfil initially for all agents as they have no knowledge about the location of all base stations. 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. Three different communication topologies are analysed in the simulation, including (1) a complete topology, where every pair of agents are neighbors; (2) a clustered topology, where the groups of Ar_i, Gw_i, Gf_i are fully connected within each group, but only Ar_1, Gw_1, Gf_1 are neighbors between different groups; (3) a dynamic topology, where each agent has a communication radius (set to 5 for all agents) as introduced in Section IV-A.

B. Simulation Results

Algorithm 6 is implemented in the experiments. For aerial vehicles the product automaton has 240 states and 10296 edges, while the ground vehicles have a product automaton with 1920 states and 63180 edges. We choose (α, γ) to be $(1000, 10)$ to construct the product automaton. We set N_k^{call} and T_k^{call} to be $(40, 20)$ for aerial vehicles and $(80, 15)$ for ground vehicles. Fig. 1, 2 show the final optimal motion plan of groups Ar_i, Gf_i, Gw_i. It can be seen that both soft and

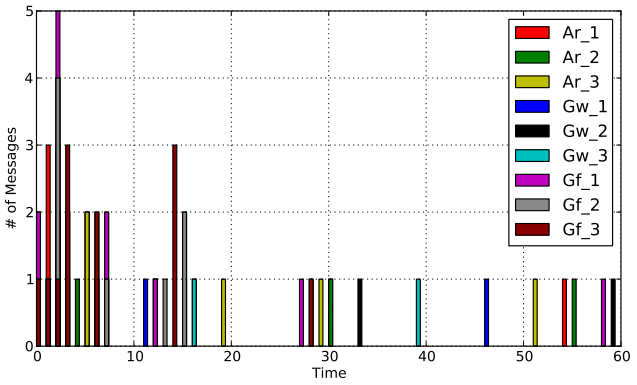

Fig. 4. This figure shows the number of messages (including sensing and reply messages) received by each agent under the clustered topology.

hard specifications are fulfilled by all agents.

Fig. 3 shows how the total costs of the aerial vehicles' motion plans evolve with time under different three communication topologies. Under the complete topology, three aerial vehicles converge to the same optimal plan within 10 steps while it takes 65 steps under the clustered topology and around 160 steps under the dynamic topology. Nevertheless, it can be seen that the optimal planner, namely Algorithm 1, is only called sparsely during the process following the triggering rule in Section V-C. Fig. 4 illustrates the number of messages received by each agent under the cluster topology. Compared with the synchronized solution that requires information exchange at each time step [10], [18], the messages are only sent and received following the subscriber-publisher scheme introduced in Section IV-A.

It is worth mentioning that in our implementation Algorithm 5 that revises the invalid or unsafe plan in average takes 0.01 seconds while Algorithm 1 that generates the optimal plan takes around 4 minutes for all agents.

VIII. Conclusion and Discussion

We have proposed a knowledge transfer scheme for cooperative motion 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 motion plans. Future work could include the consideration of dependent task specifications.

References

[1] A. Bhatia, L. E. Kavraki, M. Y. Vardi. Sampling-based motion planning with temporal goals. *IEEE International Conference on Robotics and Automation(ICRA)*, 2010.

[2] A. Bhatia, M. R. Maly, L. E. Kavraki, M. Y. Vardi. Motion planning with complex goals. *IEEE Robotics & Automation Magazine*, 18(3): 55-64, 2011.

[3] C. Baier, J.-P Katoen. Principles of model checking. *The MIT Press*, 2008.

[4] C. Belta, V. Isler, G. J. Pappas. Discrete abstractions for robot motion planning and control in polygonal environments. *IEEE Transactions on Robotics*, 21(5): 864-874, 2005.

[5] C. Belta, A. Bicchi, M. Egerstedt, E. Frazzoli, E. Klavins, G. J. Pappas. Symbolic planning and control of robot motion. *IEEE Robotics and Automation Magazine*, 14: 61-71, 2007.

[6] S. Boyd, L. Vandenberghe. *Convex Optimization*, Cambridge University Press, 2009.

[7] E. P. F. Chan, Y. Yang. Shortest Path Tree Computation in Dynamic Graphs. *IEEE Transactions on Computers*, 58(4): 541-557, 2009.

[8] E. M. Clarke, O. Grumberg, D. A. Peled. Model checking. *The MIT Press*, 1999.

[9] J. Desai, J. Ostrowski, V. Kumar. Controlling formations of multiple mobile robots. *IEEE International Conference on Robotics and Automation(ICRA)*, 2864-2869, 1998.

[10] X. Ding, M. Kloetzer, Y. Chen, C. Belta. Automatic deployment of robotic teams. *IEEE Robotics Automation Magazine*, 18: 75-86, 2011.

[11] G. E. Fainekos, A. Girard, H. Kress-Gazit, G. J. Pappas. Temporal Logic Motion Planning for Dynamic Mobile Robots. *Automatica*, 45(2): 343-352, 2009.

[12] I. F. Filippidis, D. V. Dimarogonas, K. J. Kyriakopoulos. Decentralized Multi-Agent Control from Local LTL Specifications. *IEEE Conference on Decision and Control(CDC)*, 2012.

[13] P. Gastin, D. Oddoux. Fast LTL to Büchi automaton translation. *International Conference on Computer Aided Verification (CAV'01)*, 2001.

[14] M. Guo, K. H. Johansson, D. V. Dimarogonas. Revising Motion Planning under Linear Temporal Logic Specifications in Partially Known Workspaces. *IEEE International Conference on Robotics and Automation(ICRA)*, 2013.

[15] M. Guo, D. V. Dimarogonas. Reconfiguration in Motion Planning of Single- and Multi-agent Systems under Infeasible Local LTL Specifications. *IEEE Conference on Decision and Control(CDC)*, 2013, to appear.

[16] W.P.M.H. Heemels, K. H. Johansson, and P. Tabuada. An introduction to event-triggered and self-triggered control. *IEEE Conference on Decision and Control(CDC)*, 2012.

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

[18] S. Karaman, E. Frazzoli. Vehicle routing with linear temporal logic specifications: Applications to Multi-UAV Mission Planning. *Navigation, and Control Conference in AIAA Guidance*, 2008.

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

[20] M. Kloetzer, X. C. Ding, C. Belta. Multi-robot deployment from LTL specifications with reduced communication, *IEEE Conference on Decision and Control(CDC)*, 2011

[21] D. E. Koditschek, E. Rimon. Robot navigation functions on manifolds with boundary. *Advances Appl. Math.*, 11:412-442, 1990.

[22] Y. Kuwata, J. Teo, G. Fiore, S. Karaman, E. Frazzoli. Real-time motion planning with applications to autonomous urban driving. *IEEE Transactions on Control Systems*, 17(5): 1105-1118, 2009.

[23] S. M. LaValle. Planning algorithms. *Cambridge University Press*, 2006.

[24] S. C. Livingston, R. M. Murray, J. W. Burdick. Backtracking temporal logic synthesis for uncertain environments. *IEEE International Conference on Robotics and Automation(ICRA)*, 2012.

[25] S. R. Lindemann, I. I. Hussein, S. M. LaValle. Real time feedback control for nonholonomic mobile robots with obstacles. *IEEE Conference on Decision and Control(CDC)*, 2006.

[26] M. R. Maly, M. Lahijanian, L. E. Kavraki, H. Kress-Gazit and M. Y. Vardi. Iterative temporal motion planning for hybrid systems in partially unknown environment. *In Hybrid Systems: Computation and Control (HSCC)*, 2013.

[27] J. Tumova, L. I. Reyes-Castro, S. Karaman, E. Frazzoli, and D. Rus. Minimum-violating planning with conflicting specifications. *In American Control Conference*. 2013.

[28] J. Tumova, G. Hall, S. Karaman, E. Frazzoli, and D. Rus. Least-violating strategy synthesis with safety rules. *In Hybrid Systems: Computation and Control (HSCC)*, 2013

[29] T. Wongpiromsarn, U. Topcu, R. Murray. Receding horizon temporal logic planning, *IEEE Transactions on Automatic Control*, 2012.

[30] A. Ulusoy, S. L. Smith, C. Belta. Optimal Multi-robot path planning with LTL constraints: guaranteeing correctness through synchronization. *International Symposium on Distributed Autonomous Robotic Systems*, 2012.