frontiers in Robotics and AI

ORIGINAL RESEARCH
published: 27 June 2016
doi: 10.3389/frobt.2016.00036

Resilience through Learning in Multi-Agent Cyber-Physical Systems

Konstantinos Karydis[1], Prasanna Kannappan[2], Herbert G. Tanner[2], Adam Jardine[3] and Jeffrey Heinz[3]*

[1] Department of Mechanical Engineering and Applied Mechanics, University of Pennsylvania, Philadelphia, PA, USA, [2] Department of Mechanical Engineering, University of Delaware, Newark, DE, USA, [3] Department of Linguistics and Cognitive Science, University of Delaware, Newark, DE, USA

The paper contributes to the design of secure and resilient supervisory Cyber-Physical Systems (CPS) through learning. The reported approach involves the inclusion of learning modules in each of the supervised agents, and considers a scenario where the system's coordinator privately transmits to individual agents their action plans in the form of symbolic strings. Each agent's plans belong in some particular class of (sub-regular) languages, which is identifiable in the limit from positive data. With knowledge of the class of languages their plans belong to, agents can observe their coordinator's instructions and utilize appropriate Grammatical Inference modules to identify the behavior specified for them. From that, they can then work collectively to infer the executive function of their supervisor. The paper proves that in cases where the coordinator fails, or communication to subordinates is disrupted, agents are able not only to maintain functional capacity but also to recover normalcy of operation by reconstructing their coordinator. Guaranteeing normalcy recovery in supervisory CPSs is critical in cases of a catastrophic failure or malicious attack, and is important for the design of next-generation Cyber-Physical Systems.

Keywords: multi-agent systems, leader decapitation, resilience, supervisory control, grammatical inference, cryptography

Edited by:
Andreas Kolling, The University of Sheffield, UK

Reviewed by:
Sabine Hauert, University of Bristol, UK
Jonathan M. Aitken, The University of Sheffield, UK

****Correspondence:***
Konstantinos Karydis
kkarydis@seas.upenn.edu

Specialty section:
This article was submitted to Multi-Robot Systems, a section of the journal Frontiers in Robotics and AI

Received: *15 February 2016*
Accepted: *09 June 2016*
Published: *27 June 2016*

Citation:
Karydis K, Kannappan P, Tanner HG, Jardine A and Heinz J (2016) Resilience through Learning in Multi-Agent Cyber-Physical Systems. Front. Robot. AI 3:36. doi: 10.3389/frobt.2016.00036

1. INTRODUCTION

1.1. Context and Motivation

Cyber-Physical Systems (CPSs) encompass a wide range of networked software and control units, sensors and actuators, and the intrinsic communication channels among these components. Some application areas of CPSs include automated factories, smart and energy efficient buildings, the smart grid, and self-driving cars; (Kim and Kumar, 2012; Khaitan and McCalley, 2014) provide additional application areas. As the domain of applicability of CPSs expands into security-sensitive areas, such as automated highways, water and electricity distribution systems, and military command and control, it is critical to develop novel design and control paradigms that go beyond the traditional notions of robustness, reliability, and stability (Rieger et al., 2009). These novel paradigms – or *elements of resilience* – include cyber-security and privacy [e.g., Liu et al. (2012)], control system and information network robustness, and the ability to maintain normalcy and restore function following a failure or a malicious attack.

Making a system resilient to such events is non-trivial due to the high degree of inter-connectivity among the physical and software components, and the intricate cyber, cognitive, and human

inter-dependencies (Rieger, 2014). One way to approach this problem (Rieger et al., 2009) is by decomposing it into two broad research thrusts. One of them is *State Awareness*, which is related to efficient and timely monitoring for the purpose of ensuring normalcy. The other is *Resilient Design*, which is related to enabling the system to take the appropriate control actions to maintain normalcy. This paper focuses on the latter.

Resilient design is a broad field of research that requires novel design and control paradigms and creative integration of methodological elements borrowed from various – originally disjoint – control-theoretic areas. Examples of the former category include fault-tolerant control (Blanke et al., 2003), resilient control (Mahmoud, 2004), and robust control of networked systems (Hespanha et al., 2007; Schenato et al., 2007). The latter category is a novel, rapidly expanding area of research, and some recent examples involve secure control (Cardenas et al., 2008; Mo and Sinopoli, 2009), and modeling of attacks and their impact (Kundur et al., 2011; Cam et al., 2014). Recently launched research initiatives, such as FORCEs (CPS-FORCES, 2015), aim to increase the understanding of how to design resilient large-scale, human-in-the-loop systems by combining tools from resilient control with economic incentive schemes. Yet, although resilience can emerge as a result of judicious design of interacting agent objectives and incentives, (Rieger et al., 2009) argue that, much like any organization will fail without organizational leadership, a supervisory design is still needed to ensure smooth operation. At this time, it is unclear what happens in cases where the supervisor ceases functioning, following a malicious attack or a catastrophic failure. How can one prevent the whole system from collapsing *and* recover normalcy of operation?

To answer this question, the paper proposes a different resilient design paradigm that incorporates learning to enable the system to recover normalcy of operation following a coordinator decapitation. It considers a class of supervisory systems,[1] and integrates machine learning modules based on formal languages and *Grammatical Inference* (de la Higuera, 2010) within the subordinate control units (agents). The idea is that agents keep track of the commands sent by the coordinator during normal operation, and through a particular machine learning procedure they can infer the plan(s) that the supervisor instructs them to execute, even if the agents themselves have no prior knowledge of those specific plans. The paper shows that agents can work together to ensure four key elements that prevent destabilization after coordinator decapitation: information flow, consensus-reaching ability, functional capability, and information interpretation (Jordan, 2009). Finally, it is shown how the various aspects of the coordinator's language that each agent has individually learned can be used to recover the behavior of the coordinator and, thus, maintain normalcy.

The work that originally motivated this study has been in the context of emergency response (Kendra and Wachtendorf, 2003) that documented how New York City's emergency management center was recreated on site by means of grass-root, spontaneous efforts of citizens, after it had collapsed with the twin towers. Studies on networked system resilience after leader decapitation have also been conducted within social and political sciences, focusing on counter-terrorism tactics (Jordan, 2009). The question of why not ensuring resilience after decapitation in applications mentioned (Kim and Kumar, 2012; Khaitan and McCalley, 2014) by merely maintaining a copy of the coordinating element's function in each of the subordinate agents becomes clearer if one starts considering the risks and costs related to cyber-security and privacy, in private, public, and military networks, the vulnerabilities of which to deliberate cyber-attacks are highlighted in the news (Bruni, 2014). In light of these developments, a counter-question to the one raised just above is why would anyone create multiple security liabilities by distributing copies of sensitive information and procedures.

The proposed approach complements existing ones by providing an additional feature to include in the resilient design of the next-generation CPSs. This feature is important because, in essence, equipping the agents with a learning module allows them to learn features of the organizational patterns of their supervisory system and the role assignment during normal operation. Preservation of these features following coordinator decapitation has been found to be an important factor that contributes to resilience (Kendra and Wachtendorf, 2003). Additionally, the tools developed in this work are general, and can be used in a variety of applications, such as emergency response, privacy, security and counter-intelligence, command and control, and learning through human–robot interactions. This paper extends the results obtained in Kannappan et al. (2016) by developing the theoretical foundations of this line of work, and showing that the approach is applicable to more than two subordinate agents.

1.2. Problem Description

More technically, the problem considered is the following. A system consists of a coordinator (leader), indexed by 0, and $k \in \mathbb{N}^+$ subordinate autonomous agents. The group coordinator is a processing unit that broadcasts instructions to all other members, and its messages have a structure that dictates unambiguously which agent is to execute which plan of action. A plan is understood as a temporal sequence of actions. Communication between the coordinator and each agent is private. The agents do not need to communicate with each other, but they are assumed capable to establish an all-to-all *ad hoc* network in cases of emergency in order to recover normalcy.

The group operates in an environment that imposes conditional effects upon which agent actions can be executed at any given environment state; it is important to notice here that agent actions may not necessarily change the state of the environment. The dependence between agent actions and environment states is assumed Markovian. Agent actions that are compatible with a given world state depend only on the current state, while the next environment state depends entirely on how the agents act at the current state. Furthermore, the coordinator is designed so that the generated plans can be executed by the subordinate agents; in other words, commanded agent actions are guaranteed to be implementable at the current world state.

The controlled (desired) behavior of agent $i \in \{1, \ldots, k\}$ is captured by a formal language L_i over a particular alphabet of symbols, Σ_i. The language of agent i is a set of finite combinations

[1] Examples of such systems may be scada systems or separation kernel architectures.

of letters (or symbols) from Σ_i which are called words (or strings). L_i is the *specification language* of agent i, and expresses what the agent is tasked to do. Note that the specification language of an agent is in general different from its *capacity*; the latter is a superset of L_i expressing everything that this agent is capable of doing. The plans issued by the coordinator to agent i are words that belong in L_i, however the subordinate agents do not know *a priori* what their specification language is. What is known to them is that the specification language belongs to a particular class of languages, and that this class is identifiable asymptotically from examples of elements from this language – such a language is called identifiable in the limit from positive presentations; this notion of learning is defined formally in Section 2.

There are several reasons for withholding from agents information about local objectives or the specifications, despite that the release of such information would seem to naturally endow the system with resilience properties and obviate the need for learning – agents could then operate in a completely decentralized fashion. To see this, take, for example, the common method for adding robustness (and in some sense resilience) to a distributed system: equipping every agent with a copy of the decision making algorithm. This strategy is effective if agent behaviors are *not* interdependent, in the sense that that action or inaction on behalf of one agent can *not* block execution for the remaining system. This condition holds in instances where agents swarm or flock (Tanner et al., 2007), but not when team behavior is carefully sequenced and scheduled in an orchestrated fashion as in the application context considered here.

Indeed, in the scenario considered, distributing the planning capability among the agents does not improve robustness: it is equally bad if either the coordinator or *any* of the agents fails. In fact, as the paper suggests, it is probably better if the coordinator – rather than any of the agents – fails, because the function of the former can be recovered by the latter. In addition, there may be privacy (and) or security reasons why the global strategy generation mechanism should not be proliferated across the system components: an attacker would then be able to exploit vulnerabilities at any of the distributed agent sites to gain access and insight into how the whole organization is structured and controlled. Alternatively, the strategy algorithm can be maintained in a single, remote, and secured physical device. Such distributed architecture, including physically separate, private communication channels between unsecured and trusted processes (cf. **Figure 1**), is the hallmark of separation kernels used in cryptography and secure system design (Rushby, 1981). In addition to enhancing security, it is argued (Rushby, 1981) that these distributed architectures facilitate formal verification, especially in contexts with isolated channels of different security levels (Martin et al., 2000), which become increasingly prevalent given the trend for miniaturization of communication devices.

A first question raised now is whether an agent can learn its specification language if it observes the instructions given by its coordinator for sufficiently long time. The answer to this question is straightforward and is affirmative; this is a direct consequence of known results in the field of Grammatical Inference that deals with properties of classes of formal languages and associated learning techniques (more on that in Section 2). What is not so clear is whether the agents can reconstruct the language of their coordinator once they learn their own specification language. The distinction is subtle, and it involves a certain type of synchronization that needs to occur between the strings in the individual specification languages – not every combination works. The paper shows that the answer to this question is also affirmative, and proceeds to note that these two simple facts directly bring about the seemingly surprising realization that the *mechanism* by which the coordinator devised plans for its group, although originally private, can be revealed within the organization. The agents can reconstruct faithfully the function of their coordinator, should the latter ceases to exist, through decentralized inference and inter-agent communication.

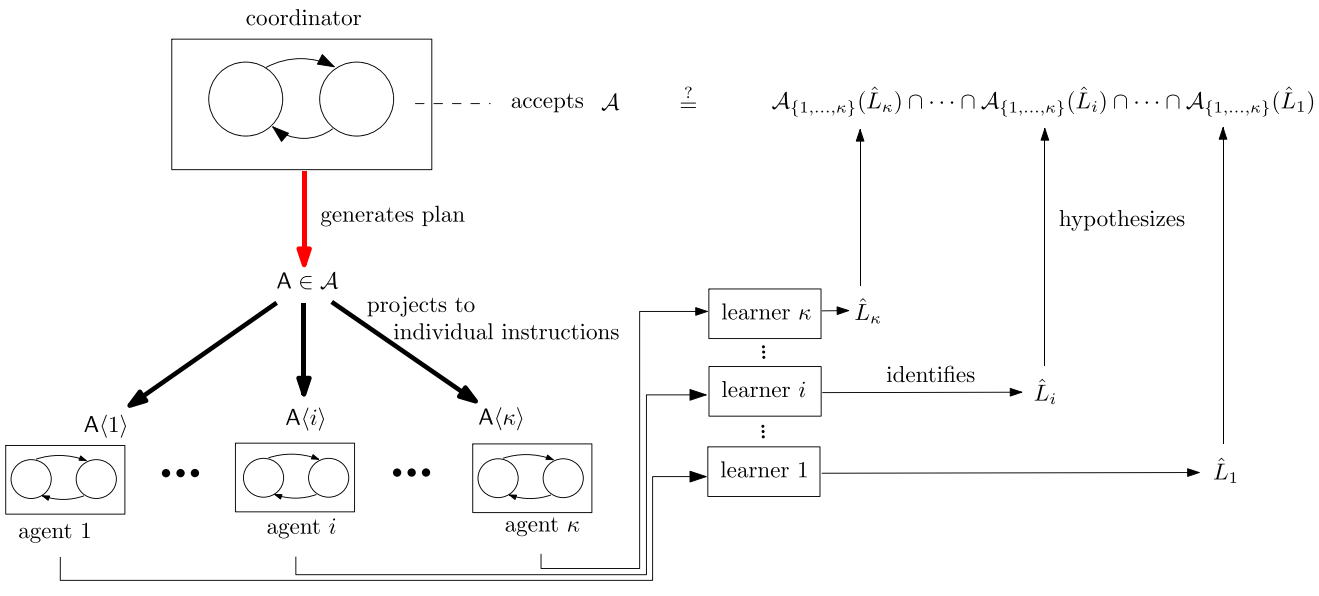

FIGURE 1 | Conceptual diagram of the supervisory system operation and learning functions in support of resilience.

1.3. Organization

The rest of the paper is organized as follows. Section 2 presents the necessary technical preliminaries on Grammatical Inference. Section 3 develops the technical machinery for learning elements of resilience in CPS, and Section 4 discusses the results and potential avenues for future research. Finally, Section 5 concludes.

2. MATERIALS AND METHODS

What follows is a brief description of formal languages and learning techniques for a class of languages we consider in this work. The introduced terminology is then used to provide a technical description of the problem tackled here.

2.1. Formal Languages

An *alphabet* is a finite set of symbols; here, alphabets are referred to with capital Greek letters (Σ or Δ). A *string* is a finite concatenation of symbols σ, taken from an alphabet Σ. In this sense, strings are "words," formed as combinations of "letters," within a finite alphabet. A string u is of the form

$$u = \sigma_0\,\sigma_1\,\sigma_2 \cdots \sigma_n \quad \text{such that each } \sigma_i \in \Sigma.$$

For a string w let $|w|$ denote its length. The *empty string* λ is the string of length 0. For two strings u, v, uv denotes their concatenation. Let Σ^* denote the set of all strings (including λ) over alphabet Σ, and Σ^n all strings of length n over Σ. For strings v, $w \in \Sigma^*$, v is a *substring* of w if and only if there exist some u_1, $u_2 \in \Sigma^*$ such that $u_1 v u_2 = w$. The *k-factors* of a string w, denoted $f_k(w)$, are its substrings of length k. Formally,

$$f_k(w) = \begin{cases} \{u \in \Sigma^k |\ u \text{ is a substring of } w\}, & \text{if } |w| \geq k \\ \{w\}, & \text{otherwise} \end{cases}.$$

Subsets of Σ^* are called stringsets, or *languages*. By default, all languages considered here are assumed to contain λ. A *grammar* is a finite representation of a (potentially infinite) language. For a grammar G, let $L(G)$ denote the language represented by G. A *class* of languages $\mathcal{L}$ is a set of languages, e.g., the set of languages describable by a particular type of grammar.

This paper will make use of the *Locally k-Testable* class of languages (McNaughton and Papert, 1971; García and Ruiz, 2004). A language L is Locally k-Testable if there is some k such that, for any two strings w, $v \in \Sigma^*$, if $f_k(w) = f_k(v)$ then either *both* w and v are in L or *neither* are. Thus, a Locally k-Testable language is one for which membership in that language is decided entirely by substrings of length k.

For example, let $\Sigma = \{a, b\}$ and L_{bb} be the set of strings over Σ, which contain at least one bb substring. In other words,

$$L_{bb} = \{bb, abb, bba, bbb, aabb, abba, abbb, \ldots\}.$$

L_{bb} is Locally 2-Testable because for any $w \in \Sigma^*$, whether or not w is a member of L_{bb} can be determined by seeing if $f_2(w)$ contains bb.

In fact, L_{bb} belongs to a *subclass* of the Locally k-Testable languages for which any language in the subclass can be described by a grammar G which is simply a *required* k-factor; i.e., $L(G) = \{w\,|G \in f_k(w)\}$. For example, L_{bb} is $L(G)$ for $G = bb$. This particular subclass is used here in the context of application examples, since its member languages can be learned from positive data in a straightforward way, as described below.

2.2. Language Identification in the Limit

The learning paradigm used in this work is that of *identification in the limit from positive data* (Gold, 1967). The particular definition here is adapted from Fu et al. (2013): given a language L, a *presentation* ϕ of L is a function ϕ: $\mathbb{N} \rightarrow L \cup \#$, where # is a symbol not in Σ, and represents a point in the text with no data. Then ϕ is a *positive presentation of* L if for all $w \in L$, there exists $n \in \mathbb{N}$ such that $\phi(n) = w$.

Let $\phi[i]$ denote the *sequence* $\phi(0)$, $\phi(1)$,. . ., $\phi(i)$. A *learner* or *grammatical inference machine* GIM is an algorithm that takes such a sequence as an input and outputs a grammar. A learner is said to *converge* on a presentation ϕ if there is some $n \in \mathbb{N}$ that for all $m > n$, $\mathsf{GIM}(\phi[n]) = \mathsf{GIM}(\phi[m])$.

A learner GIM is said to *identify a class* $\mathcal{L}$ *of languages in the limit from positive data* if and only if for all $L \in \mathcal{L}$, for all positive presentations ϕ of L, there is some point $n \in \mathbb{N}$ at which GIM converges and $L(\mathsf{GIM}(\phi[n])) = L$. Intuitively, given any language in $\mathcal{L}$, GIM can learn from some finite sequence of examples of strings in L a grammar that represents L.

This idea of learning is very general, and there are many classes of formal languages for which such learning results exist. For reviews of some of these classes, see de la Higuera (2010) and Heinz and Rogers (2013). Thus, while demonstrated with a particular subclass of the Locally k-Testable languages, the results in this paper are independent of the particular class from which the specification languages of the agents are drawn, as long as the class is identifiable in the limit from positive data.

2.3. Problem Statement

The basic problem treated in this paper is illustrated in **Figure 1**. A discrete-event system in the role of a team coordinator encodes the desired operation of its system in the form of some type of a collection of combined system trajectories $\mathcal{A}$. The coordinator generates plans A, elements of $\mathcal{A}$, which direct the behavior of a number $1,\ldots,\kappa$ of subordinate agents. Each coordinator plan A is essentially a bundle of open-loop control laws, one per agent, so each agent i receives its own control trajectory $\mathsf{A}\langle i\rangle$ through a projection operation that strips away all information that is not related to that particular agent.

In the process of executing their instructions, the agents can potentially learn incrementally the strategy behind their coordinator's plans. This paper hypothesizes that the process through which agents infer the strategy behind the instructions they receive can be implemented in a decentralized fashion by means of agent-specific inference machines (learners). Such a learner operates on the body of instructions provided to its agent up to that moment in time.

If agents can indeed figure out the rules behind the instructions they receive, a following question is whether they can combine this acquired knowledge to construct collectively a system that reproduces the behavior of their coordinator. This way, if at any

point their leader is decapitated, either due to system failure or disruption of communication, the agents are able to recover normalcy of operation by "cloning" their coordinator.

The goal of this paper is to show that the answer to both questions above, namely whether (i) agents can learn their behavior specification through observation and (ii) they can collectively reconstruct the machine that coordinates them, is affirmative. Readers familiar with results in language identification will have little reason to doubt (i), once it is revealed that system behaviors are represented as formal languages; the technical complication lies in the projection of A into $\mathsf{A}\langle\cdot\rangle$ components. Whether hypothesis (ii) is valid is less obvious and has to be treated in some more detail.

3. RESULTS

The main claim of this paper is that individual agents can gradually learn the plans that the coordinator has devised, by observing the descending commands the coordinator sends to them during the execution of a desired task. This way, if this coordinator unexpectedly fails or communication is disrupted, the agents will have learned their individual local specifications, and by combining their local hypotheses about what they are expected to do, they can essentially reconstruct an image of their lost commander.

The mathematical proof of this claim is constructive. The key to develop this proof is practically in the structure of the object types defined, and in the operations between the objects in these types.

3.1. The Models

Consider $\kappa \in \mathbb{N}^+$ agents indexed by $i \in \{1,\ldots,\kappa\}$. Their dynamics are modeled as transition systems denoted by $\mathcal{T}_i$.

Definition 1. A transition system is a tuple $\mathcal{T} = (Q, \Sigma, \rightarrow)$ with

Q	A finite set of states
Σ	A finite set of actions
$\rightarrow: Q \times \Sigma \rightarrow Q$	The transition function

Transition system $\mathcal{T}_i$ can generate every run agent i can produce – this is referred to as the *capacity* of agent i.

Definition 2. The *capacity* of agent i is a transition system $\mathcal{T}_i = (\Delta, \Sigma_i, \rightarrow_i)$ with

Δ	A finite set of (world) states
Σ_i	A finite set of actions
$\rightarrow_i: \Delta \times \Sigma_i \rightarrow_i \Delta$	The transition function

Symbols in Δ are understood as (world) states in transition system $\mathcal{T}_i$, in other words, they express the state of the world in which the agent is operating. Since the agents are operating in a common workspace and possibly interacting with each other, they are assumed to share alphabet Δ.

Transition systems can be thought of as accepting *families* of languages. However, once initial states $\Delta^I \subseteq \Delta$ and final states $\Delta^F \subseteq \Delta$ are marked on $\mathcal{T}$, the latter becomes an automaton $\boldsymbol{T}$ that accepts a particular (regular) language L. Let $\boldsymbol{T}_i$ be the automaton derived from $\mathcal{T}_i$ when *all* states are marked as both initial and final, i.e., $\Delta = \Delta^I = \Delta^F$.

In the context of this paper, the process of marking initial and final states is thought of as a product operation (Cassandras and Lafortune, 2008) between the transition system and a language specification automaton $\boldsymbol{T}_{L_i} = \langle G_i, G_i^I, G_i^F, \Sigma_i, \rightarrow_{L_i}\rangle$.

Definition 3. The *specification* of agent i is an automaton $\boldsymbol{T}_{L_i} = (G_i, G_i^I, G_i^F, \Sigma_i, \rightarrow_{L_i})$ with

G_i	A finite set of (internal) states
$G_i^I \subseteq G_i$	A finite set of initial states
$G_i^F \subseteq G_i$	A finite set of final states
Σ_i	A finite set of actions
$\rightarrow_{L_i} : G_i \times \Sigma_i \rightarrow_{L_i} G_i$	The transition function

When agent i behaves in a way consistent with its specification, the corresponding alphabet strings are exactly the input strings of the automaton $\boldsymbol{T}_{C_i} = \boldsymbol{T}_i \times \boldsymbol{T}_{L_i}$, where $\times$ denotes the standard product operation on automata (Cassandras and Lafortune, 2008).

Definition 4. The *constrained dynamics* of agent i satisfying specification $\boldsymbol{T}_{L_i}$ is an automaton

$$\boldsymbol{T}_{C_i} = (\Delta \times G_i, \Delta \times G_i^I, \Delta \times G_i^F, \Sigma_i, \rightarrow_{C_i}) . \tag{1}$$

having as components

$\Delta \times G_i$	A finite set of states
$\Delta \times G_i^I$	A finite set of initial states
$\Delta \times G_i^F$	A finite set of final states
Σ_i	A finite set of actions
$\rightarrow_{C_i} : \Delta \times G_i \times \Sigma_i \rightarrow_{C_i} \Delta \times G_i$	The transition function[a]

[a]For $\delta, \delta' \in \Delta, g, g' \in G_i$, and $\sigma \in \Sigma_i$, one has $\delta \xrightarrow{\sigma}_i \delta' \wedge g \xrightarrow{\sigma}_{L_i} g' \implies (\delta, g) \xrightarrow{\sigma}_{C_i} (\delta', g')$.

For computational expedience, the paper assumes that $\boldsymbol{T}_{L_i}$ generates a language L_i that belongs to a particular subset of Locally k-Testable class of languages (see Section 2). In this subclass, each string contains a specific k-factor. In other words, if z is the required k-factor, then any string w accepted by $\boldsymbol{T}_{L_i}$ can be written as $w = uzv$ where $u, v \in \Sigma^*$.

If ϕ is a positive presentation of L_i, and $\phi_i[m]$ is the sequence of the images of $1,\ldots,m$ under ϕ_i, then the set of factors in the string $\phi_i(r)$ associated with $r \in \{1,\ldots,m\}$ with $|\phi(r)| \geq k$ is

$$f_k(\phi_i(r)) = \{z \in \Sigma_i^k \mid \exists u, v \in \Sigma_i^* : \phi(r) = uzv\}. \tag{2}$$

The learner that identifies L_i in the limit can be compactly expressed through the equation

$$\mathsf{GIM}(\phi[m]) = \bigcap_{r=1}^{m} f_k(\phi(r)) . \tag{3}$$

Knowing that there is only one k-factor that needs to be found in all strings of L_i, one can determine when the learner has converged. Indeed, this happens when for some $m \in \mathbb{N}$, it holds that $|\mathsf{GIM}(\phi[m])| = 1$.

3.2. The Types

Let a *symbol vector* v of length κ, be defined as an ordered collection of symbols arranged in a column format, where the symbol σ_i at location i belongs to Σ_i:

$$\mathsf{v} \triangleq \begin{bmatrix} \sigma_1 \\ \sigma_2 \\ \vdots \\ \sigma_\kappa \end{bmatrix},$$

Sometimes, to save (vertical) space, v is written in the form of a row, using parentheses instead of square brackets, and separating its elements with a comma:

$$\mathsf{v} = (\sigma_1, \sigma_2, \ldots, \sigma_\kappa).$$

A concatenation of symbol vectors of the same length makes an *array*. The array has the same number of rows as the length of any vector in this concatenation. Every distinct vector concatenated forms a *column* in this array. A vector is a (trivial) array with only one column. A row in an array is understood as a string. Thus, an array can be thought of being formed, either by concatenating vectors horizontally, or by stacking (appending) strings of the same length vertically.

The notation used distinguishes vectors from arrays and strings; strings are (horizontal) sequences of symbols without delimiters, but when writing a vector in row format, its elements are separated with a comma and are enclosed in parentheses, while an array is denoted with square brackets.

Let $\mathcal{K} \subset \mathbb{N}$, and define the class $\mathcal{A}_\mathcal{K}$ of symbol arrays with $|\mathcal{K}|$ rows and $2n$, for some $n \in \mathbb{N}$ columns over the set of symbols $\Delta \cup \Sigma$. Set $\mathcal{A}_\mathcal{K}$ contains arrays of the form

$$[\mathsf{ab}]^n : \mathsf{a} = (\underbrace{\delta, \delta, \ldots, \delta}_{\kappa \text{ times}}),\ \delta \in \Delta,$$
$$\mathsf{b} = (\sigma_1, \sigma_2, \ldots, \sigma_\kappa) \in \Sigma^\kappa,\ n < \infty.$$

The set $\mathcal{K}$ will be called the *support set* of the class. The support set of a class is used to index the rows of the arrays belonging in the class. To keep track of those indices, the arrays from a particular class are annotated with the support set of this class. For example, with $\mathcal{K} = \{1, 2, \ldots, \kappa\}$, an array $\mathcal{A}_\mathcal{K} \in \mathcal{A}_\mathcal{K}$ is written as

$$\mathcal{A}_\mathcal{K} = \begin{bmatrix} \delta_1 & \sigma_{1,1} & \delta_2 & \sigma_{1,2} & \cdots & \delta_n & \sigma_{1,n} \\ \delta_1 & \sigma_{2,1} & \delta_2 & \sigma_{2,2} & \cdots & \delta_n & \sigma_{2,n} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \delta_1 & \sigma_{\kappa,1} & \delta_2 & \sigma_{\kappa,2} & \cdots & \delta_n & \sigma_{\kappa,n} \end{bmatrix}_\mathcal{K} \tag{4}$$

For $m = 1, \ldots, n$, array $\mathcal{A}_\mathcal{K}$ has every $2m+1$ column formed as a vector with the same symbol from Δ, while symbols from columns with indices equal to $2m$ for some m, are in Σ. Note that the elements in $\mathcal{K}$ need not necessarily be consecutive integers as in the example above; it is assumed, however, that they are arranged in increasing order. Each class $\mathcal{A}_\mathcal{K}$ is assumed to contain the *empty array* Λ, which is a trivial array with no columns.

To ground the concept of an array $\mathcal{A}_\mathcal{K}$ in the context of transition systems, assume, for instance, that all agents share the same state set Q and set $\Delta = Q$. Take $\sigma_{i,j} \in \Sigma_i$. Then each row of $\mathcal{A}_\mathcal{K}$ is a sequence, the subsequence of which containing the elements with even indices denotes an input word for transition system $\mathbf{T}_i$, while the subsequence containing the elements with odd indices represents the common run that all transition systems synchronously execute.

3.3. The Operations

From an automata-theoretic perspective, the basic operation needed is a (particular) product operation, which essentially implements the intersection implied in Section 1. The product operation was referred to as "particular," because it does not conform exactly to the product definition in standard literature (Cassandras and Lafortune, 2008). The reason, it does not, is because it enforces synchronization on a component of the state of the factors, rather than their actions. This special product operation is referred to as the *synchronized product*.

To see how it works, consider two agent constrained dynamics $\boldsymbol{T}_{C_1}$ and $\boldsymbol{T}_{C_2}$, respectively, that share the same space Q as the first component of their state space. Recall the standard Trim operation on automata (Cassandras and Lafortune, 2008), which simplifies the system by retaining only its accessible[2] and co-accessible[3] states and define the synchronized product of $\boldsymbol{T}_{L_1}$ and $\boldsymbol{T}_{L_2}$ as follows.

Definition 5. The *synchronized product* $\otimes$ of $\boldsymbol{T}_{C_1}$ and $\boldsymbol{T}_{C_2}$ is defined as the operation that yields a third automaton

$$\boldsymbol{T}_{C_{1\otimes 2}} := \boldsymbol{T}_{C_1} \otimes \boldsymbol{T}_{C_2} := \mathrm{Trim}((Q \times G_1 \times G_2,\ Q \times G_1^I \times G_2^I,\ Q \times G_1^F \times G_2^F,\ \Sigma_1 \times \Sigma_2,\ \rightarrow_{C_{1\otimes 2}})) \tag{5}$$

where the transition function $\rightarrow_{C_{1\otimes 2}}$ is defined as a map $Q \times G_1 \times G_2 \times \Sigma_1 \times \Sigma_2 \to Q \times G_1 \times G_2$ where $(q, g_1, g_2) \xrightarrow{(\sigma_1,\sigma_2)}_{C_{1\otimes 2}} = (q', g_1, g_2)$ if for $q \in q, g_1, g_1' \in G_1, g_2, g_2' \in G_2, \sigma_1 \in \Sigma_1$, and $\sigma_2 \in \Sigma 2$, it is $(q, g_1) \xrightarrow{\sigma_1}_{C_1} (q', g_1')$ and $(q, g_2) \xrightarrow{\sigma_2}_{C_2} (q', g_2')$.

The operation is extended inductively to more than two factors:

$$\boldsymbol{T}_{C_1} \otimes \boldsymbol{T}_{C_2} \otimes \boldsymbol{T}_{C_3} \otimes \cdots \otimes \boldsymbol{T}_{C_n} := (\cdots((\boldsymbol{T}_{C_1} \otimes \boldsymbol{T}_{C_2}) \otimes \boldsymbol{T}_{C_3}) \otimes \cdots \otimes \boldsymbol{T}_{C_n}).$$

The effect of the synchronized product on two automata is a particular form of operation on the languages they generate. This operation is neither a pure union, nor a pure intersection. The effect is clearer when the words in the languages of each system are viewed as rows of a symbol array as in equation (4), that is, sequences in $(\Delta \times \Sigma)^*$. In fact, a string in $(\Delta \times \Sigma)^*$ can be thought of as a (trivial) $1 \times 2n$ array. The synchronized product operation now essentially *merges* these one-row arrays into a two-row array, in a way that ensures that odd columns are vectors consisted of the same symbol.

To see how this works in general with arrays, define first a *projection* operation on strings. A string u formed with symbols in an alphabet Σ can be projected to a set $\Sigma' \subset \Sigma$, by "deleting" all symbols in the string that do not belong in Σ'. The projection to Σ' operation is denoted $\pi_{\Sigma'}$ and formally defined as

$$\pi_{\Sigma'} : \Sigma^* \to \Sigma'^*; \quad u \mapsto \begin{cases} \lambda & u = \lambda \\ \pi_{\Sigma'}(s) & u = s\,a,\ a \notin \Sigma' \\ \pi_{\Sigma'}(s)a & u = s\,a,\ a \in \Sigma' \end{cases}.$$

The projection operation is extended from strings to arrays implicitly, through a function that extracts a particular row from an array. This *extraction* operation is first defined on vectors, and then naturally extended to arrays. The process is as follows. Let $\cdot[j]$ denote the (extraction) operation on vectors that selects

[2] Accessible states are all states that are reachable from initial states.

[3] Co-accessible states are states from which there exists a path to a final state.

the element (a symbol) of the vector at position $j \in \{1, \ldots, n\}$, that is,

$$\cdot\langle j\rangle : \Sigma_1 \times \Sigma_2 \times \cdots \times \Sigma_j \times \cdots \times \Sigma_n \to \Sigma_j;$$
$$\Lambda \neq u = (\sigma_1, \sigma_2, \ldots, \sigma_j, \ldots, \sigma_n) \mapsto \sigma_j.$$

The extraction operation can be naturally extended to arrays. Without loss of generality, assume that $|\mathcal{K}| = \kappa$, and arrange the elements of $\mathcal{K}$ in increasing order: $\{n_1, \ldots, n_\kappa\}$. In this case, the $\cdot\langle j\rangle$ operation yields the row (string) of the array indexed by $n_j \in \mathcal{K}$:

$$\cdot\langle j\rangle : (\Sigma_{n_1} \times \cdots \times \Sigma_j \times \cdots \times \Sigma_{n_\kappa})^* \to \Sigma_{n_j}^* \ ;$$

$$\mathsf{A}_{\mathcal{K}\times n} \mapsto \begin{cases} \lambda & \mathsf{A}_{\mathcal{K}\times n} = \Lambda \\ \lambda & n_j \notin \mathcal{K} \\ \mathsf{B}_{\mathcal{K}\times(n-1)}\langle j\rangle \mathsf{b}\langle j\rangle & \mathsf{A}_{\mathcal{K}\times n} = [\mathsf{B}_{\mathcal{K}\times(n-1)}\ \mathsf{b}]\ , \\ & \mathsf{b} \in \Sigma_{n_1} \times \cdots \times \Sigma_{n_\kappa}\ . \end{cases}$$

Let now $\mathcal{A}_{\mathcal{K}\times n}$ denote specifically the class of symbol arrays with support set[4] $\mathcal{K} \subset \mathbb{N}$ and dimension $|\mathcal{K}| \times 2n$. Consider two array classes, $\mathcal{A}_{\mathcal{I}\times n}$ and $\mathcal{A}_{\mathcal{J}\times n}$, that have the same row length $2n$, and non-intersecting support sets $\mathcal{I} \cap \mathcal{J} = \varnothing$. A *merge* operation can be defined on those two arrays in the following way:

$$\mathcal{A}_{\mathcal{I}\times n} \oplus \mathcal{A}_{\mathcal{J}\times n} \to \mathcal{A}_{\mathcal{I}\cup\mathcal{J}\times n};$$
$$\mathsf{A}_{\mathcal{I}\times n} \oplus \mathsf{A}_{\mathcal{J}\times n} \mapsto$$
$$\begin{cases} \Lambda & ;\ \mathcal{I} \cap \mathcal{J} \neq \varnothing \\ \mathsf{A}: \begin{cases} \mathsf{A}_{\mathcal{I}\cup\mathcal{J}\times n}\langle j\rangle = \mathsf{A}_{\mathcal{I}\times n}\langle j\rangle & j \in \mathcal{I} \\ \mathsf{A}_{\mathcal{I}\cup\mathcal{J}\times n}\langle j\rangle = \mathsf{A}_{\mathcal{J}\times n}\langle j\rangle & j \in \mathcal{J} \end{cases} & ;\ \pi_\Delta\big(\mathsf{A}_{\mathcal{I}\times n}\langle j\rangle\big) = \pi_\Delta\big(\mathsf{A}_{\mathcal{J}\times n}\langle j\rangle\big) \\ \Lambda & ;\ \pi_\Delta\big(\mathsf{A}_{\mathcal{I}\times n}\langle j\rangle\big) \neq \pi_\Delta\big(\mathsf{A}_{\mathcal{J}\times n}\langle j\rangle\big) \end{cases} \tag{6}$$

Notice the projection operation that checks whether the symbols in Δ match in the rows of the arrays being merged. A close comparison of equation (6) with the definition of the transition function of equation (5) reveals the equivalence. The merge operation essentially translates the synchronized product operation on automata to the corresponding operation on language arrays.

To express the fact that the substrings formed by taking every symbol with even index (i.e., not in Δ) in a row $i \in \{1, \ldots, \kappa\}$ of an array belong in a specific language $L_i \subset \Sigma_i^*$, the array class is written $\mathcal{A}_{\mathcal{K}\times n}(\{L_i\}_{i\in\mathcal{K}})$. When not all rows contain substrings in specific languages, but rather only the rows with indices belonging in $\mathcal{M} \subset \mathcal{K}$, then this can be denoted $\mathcal{A}_{\mathcal{K}\times n}(\{L_i\}_{\mathcal{M}})$ (short and for $\mathcal{A}_{\mathcal{K}\times n}(\{L_i\}_{\mathcal{M}} \cup \{\Sigma_j^*\}_{j\in\mathcal{K}\setminus\mathcal{M}})$).

3.4. The Result

Let $\mathcal{K} = \{1, \ldots, \kappa\}$ and consider κ agents, the capacity of which is modeled originally in the form of transition systems $\mathcal{T}_1, \ldots, \mathcal{T}_\kappa$. A run on $\mathcal{T}_i$ is a row-vector, just like a row in some $\mathsf{A}_\mathcal{K}$ as in equation (4), where the alphabet symbols in Σ_i are interspersed with symbols from an alphabet Δ.

The closed-loop (controlled) behavior of agent i is supposed to satisfy specification $\boldsymbol{T}_{L_i}$. The closed-loop system, which is consistent with the specification, is $\boldsymbol{T}_{C_i}$. However, since agents are not supposed to have knowledge of L_i (or, equivalently, $\boldsymbol{T}_{L_i}$), the product operation yielding $\boldsymbol{T}_{C_i}$ cannot be performed locally by each agent. Instead, the agents are "teleoperated" by a *coordinator*, a central automaton $\boldsymbol{T}_0$ that dictates specifically what transition each agent is to take at each given state. In some sense, all such product operations have been performed by this *coordinator*, a model of which can be thought of as

$$\boldsymbol{T}_0 = \boldsymbol{T}_{C_1} \otimes \cdots \otimes \boldsymbol{T}_{C_\kappa}.$$

Definition 6. The *coordinator* is an automaton

$$\boldsymbol{T}_0 = (\Delta \times G_1 \times \cdots \times G_\kappa,\ \Delta \times G_1^I \times \cdots \times G_\kappa^I$$
$$\Delta \times G_1^F \times \cdots \times G_\kappa^F, \Sigma_1 \times \cdots \times \Sigma_\kappa,\ \to)$$

with components

$\Delta \times G_1 \times \cdots \times G_\kappa$	A finite set of states
$\Delta \times G_1^I \times \cdots \times G_\kappa^I$	A finite set of initial states[a]
$\Delta \times G_1^F \times \cdots \times G_\kappa^F$	A finite set of final states[b]
$\Sigma_1 \times \cdots \times \Sigma_\kappa$	A finite set of action profiles[c]
$\to: \Delta\times G_1\times\cdots\times G_\kappa\times\Sigma_1\times\cdots\times\Sigma_\kappa \to \Delta\times G_1\times\cdots\times G_\kappa$	The transition function[d]

[a] $G_i^I \subseteq G_i$.
[b] $G_i^F \subseteq G_i$.
[c] An action profile is a tuple of symbols from agents' alphabets.
[d] For $\delta_i, \delta_j \in \Delta$, a transition $\delta_i \xrightarrow{(\sigma_1,\ldots,\sigma_\kappa)\delta_j}$ occurs if $(\delta_i, \sigma_k) \in \to_k$ for all $\kappa \in \mathcal{K}$.

A coordinator's run is a finite sequence of the form

$$r_\mathcal{K} := \overbrace{\underbrace{\delta_1}_{\text{row }1}\ g_{1,1}}^{\text{row }\kappa} \cdots \overbrace{g_{\kappa,1}}\ \underbrace{\sigma_{1,1}} \cdots \overbrace{\sigma_{\kappa,1}}\ \underbrace{\delta_2\ g_{1,2}} \cdots \overbrace{g_{\kappa,2}}\ \underbrace{\sigma_{1,2}}$$
$$\cdots \overbrace{\sigma_{\kappa,2}}\ \underbrace{\delta_n\ g_{1,n}} \cdots \overbrace{g_{\kappa,n}}\ \underbrace{\sigma_{1,n}} \cdots \overbrace{\sigma_{\kappa,n}} \cdots \tag{7}$$

where the braces indicate how the sequence elements can be regrouped with some minimal redundancy, and rearranged in the form of an array like equation (4)

$$\mathsf{A}_\mathcal{K} = \begin{bmatrix} \delta_1\, g_{1,1} & \sigma_{1,1} & \delta_2\, g_{1,2} & \sigma_{1,2} & \cdots & \delta_n\, g_{1,n} & \sigma_{1,n} \\ \delta_1\, g_{2,1} & \sigma_{2,1} & \delta_2\, g_{2,2} & \sigma_{2,2} & \cdots & \delta_n\, g_{2,n} & \sigma_{2,n} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \delta_1\, g_{\kappa,1} & \sigma_{\kappa,1} & \delta_2\, g_{\kappa,2} & \sigma_{\kappa,2} & \cdots & \delta_n\, g_{\kappa,n} & \sigma_{\kappa,n} \end{bmatrix}, \tag{8}$$

from which the link between the synchronized product on automata and the merge operation on symbol arrays is verified. Coordinator runs, either in the form of a sequence [equation (7)]

[4] Note that K may not necessarily contain consecutive integers.

or in the form of an array [equation (8)], are referred to as *plans*.

Viewing now the coordinator as the synchronized product of the *constrained* dynamics of agents, the details of communication between this coordinator and its subordinate agents can be formalized. Assume that there exists a dedicated communication channel to each agent, and an encoder that takes the row of $\mathsf{A}_{\mathcal{K}}$ that corresponds to the particular agent and extracts the sequence of input strings for that agent. Specifically, assume that the coordinator communicates $\pi_{\Sigma_i}(\mathsf{A}_{\mathcal{K}}\langle i\rangle)$ to agent i. (This can be done either in one batch, or one symbol at a time.) Then agent i executes the specified sequence of input symbols synchronously with the other agents, and *all* agents transition together through world states δ_1, δ_2,..., until some final world state δ_{n+1} (not shown in $\mathsf{A}_{\mathcal{K}}$).

Assume now that every $L_i \ni \pi_{\Sigma_i}(\mathsf{A}_{\mathcal{K}}\langle i\rangle)$ belongs to a subclass of Locally 2-Testable languages with grammars G_i consisted of a single 2-factor, i.e., $G_i = \{\{\sigma_m\sigma_k\}\}$ for σ_m, σ_k, $\in \Sigma_i$, and that each agent knows that this is the subclass of languages containing its specification. Then a GIM_i can be constructed (García and Ruiz, 2004) to identify L_i in the limit from positive data. Each plan communicated by the coordinator to the agents constitutes a positive datum, and if enough[5] data are presented to GIM_i, the learner will converge to L_i in finite time.

Imagine a moment in time when the hypothesis (output) of every GIM_i has converged to the corresponding specification language L_i. The question now is: can the agents, having knowledge of their own specification, *reconstruct* $\boldsymbol{T}_0$ by communicating? The sequence of mathematical statements that follow provide an affirmative answer to this question.

Consider a sequence $\{\mathsf{A}_{\mathcal{K}\times n}(k)\}_{k=0}^{\infty}$ of $2n$-column symbol arrays of the form [equation (8)]. Pick an arbitrary $i \in \{1, \ldots, \kappa\}$. Let the presentation to learner GIM_i of agent i be

$$\phi_i := \underbrace{\pi_{\Sigma_i}(\mathsf{A}_{\mathcal{K}\times n}(0)\langle i\rangle)}_{\phi_i(0)}, \underbrace{\pi_{\Sigma_i}(\mathsf{A}_{\mathcal{K}\times n}(1)\langle i\rangle)}_{\phi_i(1)}, \underbrace{\pi_{\Sigma_i}(\mathsf{A}_{\mathcal{K}\times n}(2)\langle i\rangle)}_{\phi_i(2)}, \ldots .$$

As assumed, there is a finite $m \in \mathbb{N}$ such that $L(\mathsf{GIM}_i(\phi_i[m])) = L_i$. At this point, and without any additional information about its teammates, an agent generically hypothesizes that the language of the coordinator is $\mathcal{A}_{\mathcal{K}\times n}(\{L_i\})$, i.e., an $\kappa \times 2n$ array class where the rows in row i are words accepted by $\boldsymbol{T}_{C_i}$, and in any row $j \neq i$ one finds any combination of symbols in Σ_j interspersed with the (same) symbols from Δ that appear in the i row.

The following lemma indicates that if two agents intersect the array classes they each hypothesize as the coordinator's language, they obtain exactly what they would have learned if they had been observing each other's presentation and running two learners in parallel, one for each presentation.

Lemma 1. $\mathcal{A}_{\mathcal{I}\cup\mathcal{J}\times n}(\{L_i\}_{i\in\mathcal{I}}) \cap \mathcal{A}_{\mathcal{I}\cup\mathcal{J}\times n}(\{L_j\}_{j\in\mathcal{J}}) = \mathcal{A}_{\mathcal{I}\cup\mathcal{J}\times n}(\{L_k\}_{k\in\mathcal{I}\cup\mathcal{J}})$.

Proof.

$$\begin{aligned}
&\mathcal{A}_{\mathcal{I}\cup\mathcal{J}\times n}\big(\{L_i\}_{i\in\mathcal{I}}\big) \cap \mathcal{A}_{\mathcal{I}\cup\mathcal{J}\times n}\big(\{L_j\}_{j\in\mathcal{J}}\big)\\
&\quad= \mathcal{A}_{\mathcal{I}\cup\mathcal{J}\times n}\Big(\{L_i\}_{i\in\mathcal{I}} \cup \{\Sigma_j^*\}_{j\in\mathcal{J}}\Big)\\
&\qquad\cap \mathcal{A}_{\mathcal{I}\cup\mathcal{J}\times n}\Big(\{L_j\}_{j\in\mathcal{J}} \cup \{\Sigma_i^*\}_{i\in\mathcal{I}}\Big)\\
&\quad= \mathcal{A}_{\mathcal{I}\cup\mathcal{J}\times n}\Big(\{L_i\cap\Sigma_i^*\}_{i\in\mathcal{I}} \cup \{L_j\cap\Sigma_j^*\}_{j\in\mathcal{J}}\Big)\\
&\quad= \mathcal{A}_{\mathcal{I}\cup\mathcal{J}\times n}\Big(\{L_i\}_{i\in\mathcal{I}} \cap \{L_j\}_{j\in\mathcal{J}}\Big)\\
&\quad= \mathcal{A}_{\mathcal{I}\cup\mathcal{J}\times n}\Big(\{L_k\}_{k\in\mathcal{I}\cup\mathcal{J}}\Big).
\end{aligned}$$

It is now natural to take this argument one step further to conclude that if all κ agents intersect their hypotheses obtained after their individual learners have converged, the resulting array class would be identical to the one that a single GIM would produce if it were to operate on a presentation of symbol arrays coming out of the coordinator.

Lemma 2. $\bigcap_{i\in\mathcal{K}} \mathcal{A}_{\mathcal{K}\times n}(L_i) = \mathcal{A}_{\mathcal{K}\times n}(\{L_i\}_{i\in\mathcal{K}})$.

Proof. Straightforward induction on i.

In this light, the intersection performed at the end, compensates for the decentralized agent operation and inference.

Proposition 1. Assume that for each $i \in \mathcal{K}$, a grammatical inference module running on inputs $\pi_{\Sigma_i}(\mathsf{A}[m]\langle i\rangle)$ has converged on a language L_i for large enough $m \in \mathbb{N}$. Then the array class where the modules' presentation has been drawn from is exactly $\bigcap_{i\in\mathcal{K}} \mathcal{A}_{\mathcal{K}\times n}(\{L_i\}_{i\in\mathcal{K}})$.

Proof. A direct restatement of Lemma 2 in the context of κ grammatical inference modules running in parallel on the rows of the finite sequence of symbol arrays $\{\mathcal{A}_{\mathcal{K}\times n}(k)\}_{k=0}^{m}$.

One interpretation of Proposition 1, therefore, is that if the agents' link to their coordinator is severed, yet they had identified their own specification before this happened, then they can resurrect their coordinator's function by intersecting their individual hypotheses $\mathcal{A}_{\mathcal{K}\times n}(\{L_i\})$ through communication.

4. DISCUSSION

4.1. Implementation Study

Consider two agents, with capacities $\mathcal{T}_1$ and $\mathcal{T}_2$; the transition systems of the capacities of the two agents are illustrated in **Figure 2**. The two agents share the same (discrete) world state set $W = \{w_0, w_1, w_2\}$. Agent has alphabet $\Sigma_1 = \{s_{10}, s_{11}, s_{12}\}$, while agent 2 has alphabet $\Sigma 2 = \{s_{20}, s_{21}, s_{22}\}$. Both agents are supervised by coordinator $\boldsymbol{T}_0$, which determines the desired behavior its subordinates. The desired behavior for an agent is its language specification, and is encoded as an automaton: T_{L_1} for agent 1, and T_{L_2} for agent 2, and shown in **Figure 3**. The labels on each specification automaton's states are (almost) arbitrary integers: the only consideration in the assignment is so that the states of the two automata can be distinguished. Here, let $G_1 = \{g_{11}, g_{12}, g_{13}\} = \{1, 2, 0\}$ and $G_2 = \{g_{21}, g_{22}, g_{23}\} = \{4, 5, 3\}$. The languages generated by T_{L_1} and T_{L_2} belong to the specific subclass of Locally 2-Testable languages considered: the specification language for agent 1 contains all strings that have $s_{12}s_{11}$ as a substring, while that for agent 2 includes all strings that have the factor $s_{22}s_{21}$.

[5]For the particular language subclass, a handful of positive examples generally suffice.

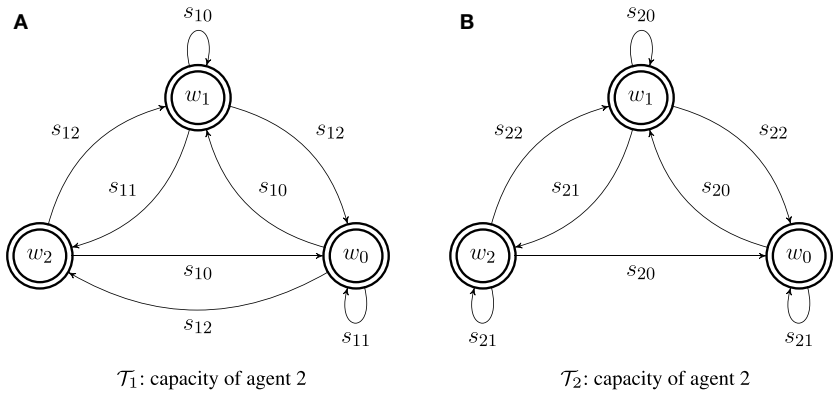

FIGURE 2 | The capacity of agents $\mathcal{T}_1$ and $\mathcal{T}_2$ shown in (A,B), respectively. Since, in a transition system, all states can be thought of as both initial and final, they are marked in the figures using circles drawn with double thick line.

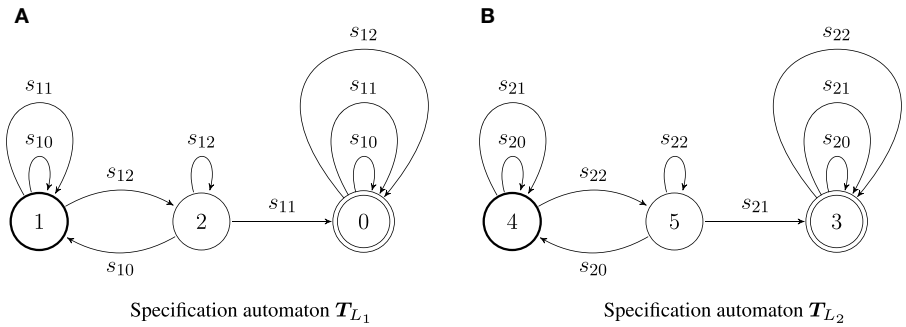

FIGURE 3 | Automata T_{L_1} (A) and T_{L_2} (B) encode the specifications for agents 1 and 2, respectively. Thick single circles denote initial states; double circles denote final states. Input strings for agent 1 belong to the specification language if they contain the factor $s_{12}s_{11}$. Input strings for agent 2 are consistent with that agent's specification if they contain the factor $s_{22}s_{21}$.

Taking the product of the agent's capacity $\mathcal{T}_i$ with its specification T_{L_i} produces the constrained dynamics of the agent, T_{C_i}. The result of the product operation for the systems depicted in **Figures 2** and **3** is shown in **Figure 4**. The coordinator $\boldsymbol{T}_0$ is formed by taking the synchronized product of T_{C_1} and T_{C_1}, which is shown in **Figure 5**. It may be worth noting that the product operation between the agent's capacity and its specification creates a *unique perspective* of a world state, from the point of view of the individual agent: for example, world state w_1 may have different semantics for agent 1 compared to agent 2, for the two agents are trying to achieve different things. Yet, as the whole group operates in the same physical workspace, agents synchronize on their world state when they act together, as shown in the synchronized product of **Figure 5**.

A run in the coordinator is now a plan for the subordinates. After translating tuple labels into strings (dropping parentheses and commas), this plan takes the form as in equation (7). One example can be:

$$r_{\{1,2\}} = w_1\,1\,4\;\; s_{10}s_{20}\;\; w_1\,1\,4\;\; s_{12}s_{22}\;\; w_0\,2\,5\;\; s_{11}s_{21}\;\; w_0\,0\,3,$$

which in tabulated form [as in equation (8)] looks like

$$\mathsf{A}_{\{1,2\}} = \begin{bmatrix} w_1\,1\,4 & s_{10} & w_1\,1\,4 & s_{12} & w_0\,2\,5 & s_{11} \\ w_1\,1\,4 & s_{20} & w_1\,1\,4 & s_{22} & w_0\,1\,5 & s_{21} \end{bmatrix}.$$

Plans are then communicated to the agents. Agent i receives $\pi_{\Sigma_i}(\mathsf{A}_{\{1,2\}}\langle i\rangle)$, and all agents execute their instructions in step (i.e., synchronously), transitioning from one common world state to a next. There can be several instances where the same task needs to be completed, and for each one of these instances the coordinator crafts a different plan – the system may be initialized at a different world state each time, or several scheduling strategies can be tried out; there is usually more than one way to achieve the same end result involving the same crucial steps. Each such instance of guided task completion offers an example for the agent learning algorithms. **Figures 6A,B** display results from several different experiments, during which the two agents observe the instructions received over a number of instances, and attempt to identify their specification. Depending on the composition of these sequences of examples, agents may need more time to identify their specification language. The two figures show the size (cardinality) of the hypothesized grammar produced by the inference machine, $|\mathsf{GIM}|$, for the specification language of agent 1 (**Figure 6A**) and agent 2 (**Figure 6B**) as a function of the number m of examples in the presentation ϕ provided to their inference algorithms by their coordinator, over a number of different experiments. Every experiment consists of an initial fragment of some (specification language) presentation, which for the purpose of these numerical tests is generated randomly.

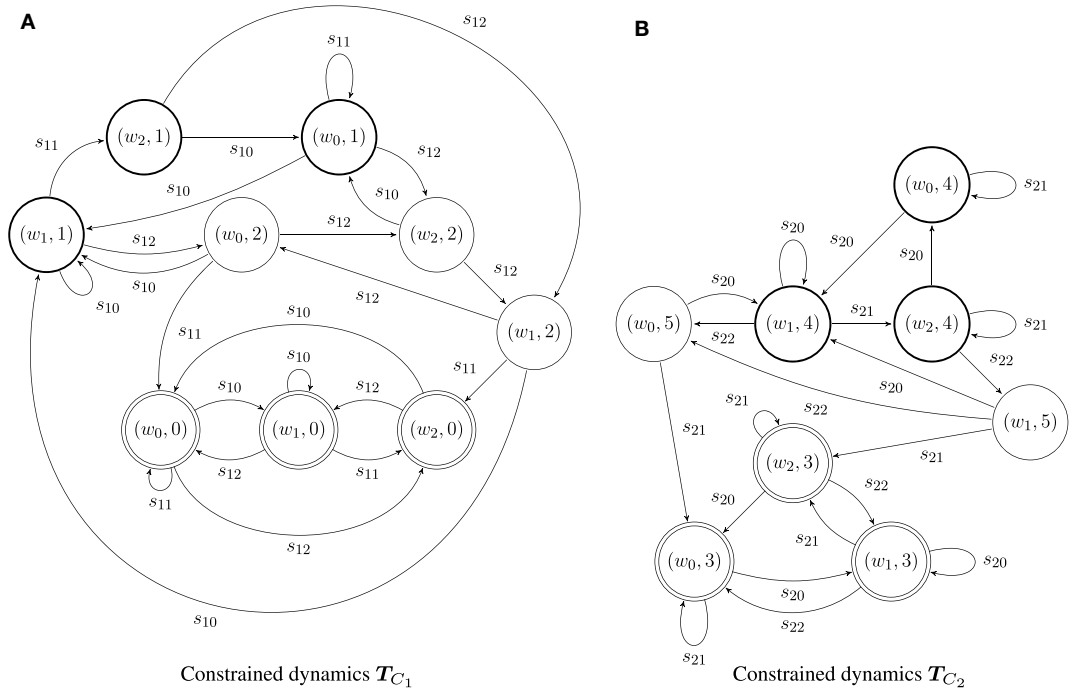

FIGURE 4 | The constrained dynamics of agents 1 and 2, (A) $T_{C_1} = \mathcal{T}_1 \times T_{L_1}$ and (B) $T_{C_2} = \mathcal{T}_2 \times T_{L_2}$, respectively.

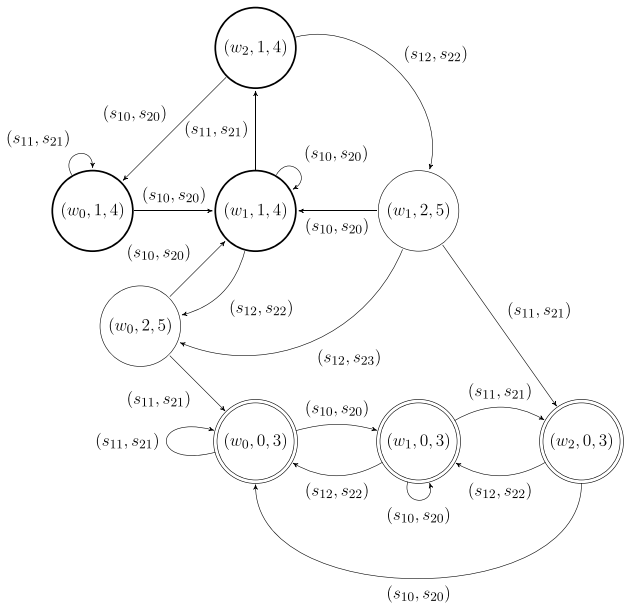

FIGURE 5 | The automaton T_0 of the coordinator. It is produced as $T_0 = T_{C_{1\otimes 2}} = T_{C_1} \otimes T_{C_2}$.

Convergence is achieved when $|\mathsf{GIM}(\phi[m])| = 1$. The thick curves in **Figures 6A,B** correspond to experimental averages of grammar size for random presentation fragments of a certain length. It appears that a (uniformly) random generation of presentations for the target Locally 2-Testable languages results in (at least) polynomial rate of convergence.

4.2. Scaling Up

Obviously, due to the product operations involved, increasing the number of agents κ, or the size of the agents' automata has an adverse effect on one's ability to reproduce the results of the previous section. Several observations, however, seem to indicate that the key insight behind the reported method is not directly linked to computational complexity issues related to dimensionality. This section, thus, briefly illustrates the implementation of the reported method on a similar setup with three agents, and concludes with the observations that can guide further algorithm development in the direction of handling larger-scale problems.

In this setup, there are three agents with structure similar to that shown in **Figure 2**. The three agents have similar but not identical capacities. There are still three world states, three-symbol alphabets for each agent, and the agents' specification languages are again Locally 2-Testable languages, each represented by an automaton with three states. (The actual automaton for the coordinator is too large to display on these pages.)

Similar to Section 1, the learning process is repeated several times, and in each trial initial segments $\phi[m]$ of random presentations of length $m = 100$ are generated and processed by the individual agent learners. Each segment is tabulated in an array $\mathsf{A}_{\{1,2,3\}}$, from which each agent reads the (projected) row $\pi_{\Sigma_i}(\mathsf{A}_{\{1,2,3\}}\langle i\rangle)$ associated with its index $i \in \{1, 2, 3\}$. As each learner reads this presentation segment, it progressively refines its hypothesis about what its specification language might be, and similarly to **Figure 6**, in **Figure 7** the number of factors in the hypothesized agent grammar is recorded in the plots of **Figures 7A–C**. **Figure 7C** seems to also indicate a polynomial rate of convergence.

The simulation of the three-agent example was coded in python and was run on a Intel-I7 Quad-core laptop computer

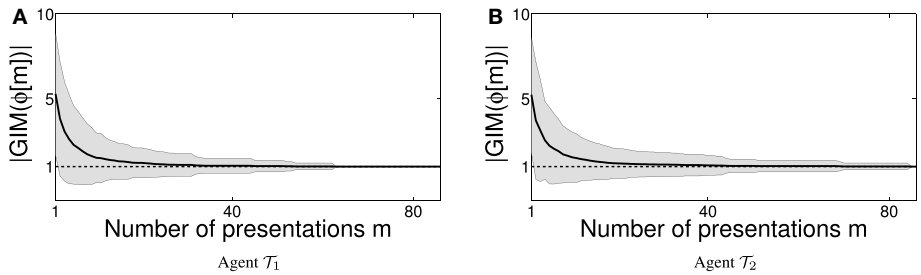

FIGURE 6 | Convergence of the learner as the number of examples *m* increases. The shaded envelopes in **(A,B)** represent the SD around the mean size of the hypothesized grammars for agents 1 and 2, indicating that |GIM($\phi[m]$)| converges to the value of 1 (marked with a dotted black line) as the number of presentations *m* increases.

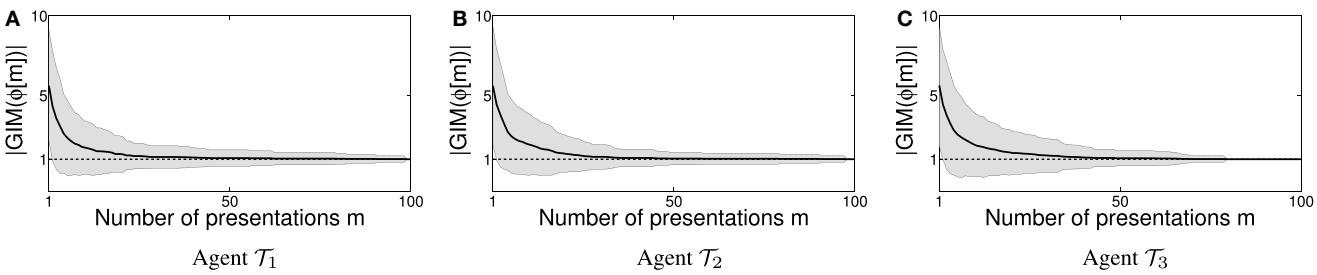

FIGURE 7 | Convergence results for individual agent learners over initial segments of presentation with 100 examples. The plots **(A–C)** show the evolution of the cardinality of each agent's hypothesized grammars; when these grammars are reduced to a single factor, then the learner has converged. The shaded envelopes indicate the evolution of the SD around the mean grammar size over all trials, as the size of the presentation to the learners increases.

(8 threads, 2.30 GHz processor). What needs to be noted, however, is that the computationally challenging aspect of such an implementation, as the number of agents and the size of their automata increase, is in computing and representing their products (with their specifications, and eventually with each other). In practice, however, this computation would be needed for determining coordination plans, not for inferring the agents' specifications. Note that if the (big) automaton that the coordinator needs to devise its plans for its subordinates is available, and the system is running in normal operation mode, then the inference algorithm on each agent's hardware would need to build and refine a machine of size independent of the number of agents in the group: each agent is learning its own specification. Only when the agents are called to *combine* their hypotheses into a single model, and construct the synchronized product, would the increased computational power be required.

Skeptics will argue that outside the realm of academic examples, one inevitably has to face analysis of many complex sub-systems (agents), and a daunting computation of a huge synchronized product will be unavoidable. It is conjectured, however, that there are computationally efficient alternatives to performing this operation. For example, once all agent constrained dynamics have been reconstructed, compatible coordinator plans can possibly be synthesized in a factored fashion, by synchronizing the runs on individual constrained dynamics incrementally, transition by transition. Preliminary evidence that supports this hypothesis is that this type of factored synthesis has been already demonstrated when constructing winning strategies in two-player zero-sum games (Fu et al., 2015), while at the same time it has been formally proven that learning sub-regular languages in factored form is also feasible (Heinz and Rogers, 2013). In fact, the identification of individual agent specifications as implemented in this paper is a manifestation of the ability to learn in factored form. Exploiting the factored structure of the system in this way allows for exponentially smaller system representations (Heinz and Rogers, 2013), significantly alleviating the ramifications of the curse of dimensionality. Exploring further the possibility for synthesis of coordinator plans when a model of the latter is maintained in factored form deserves treatment in a separate paper.

5. CONCLUSION

Distributed multi-agent systems, in which individual agents are coordinated by a central control authority, and the dynamics of all entities is captured in the form of transition systems, can be made resilient to leader decapitation by means of learning. Specifically, grammatical inference algorithms running locally at each agent can be utilized to decode the logic behind the generation of commands that are issued to each individual agent over a period of time, provided that a sufficiently large sample of command examples are observed, and the agents know *a priori* the class of formal languages their specifications belong to. Once individual agent specifications are identified, then it is shown that agents can put together their hypotheses about how their coordinator has been generating their instructions, and in this way essentially reconstruct it. This type of result can contribute to theory that supports the design of resilient multi-agent supervisory control systems, but also be utilized from the opposite direction as a means of decoding the mechanism that generates a bundle of signals communicated over a number of different, isolated, channels.

AUTHOR CONTRIBUTIONS

KK contributed primarily in exploring the related literature, and together with HT he took leadership in formulating the problem, as well as designing the overall architecture depicted in **Figure 1**. PK was responsible for generating and presenting the numerical data that support the theoretical predictions, and helped revising the presentation of the theoretical analysis in order to be in concert with the software implementation. AJ and JH led the design of the grammatical inference algorithms, and contributed to the design of numerical tests. All authors contributed to varying degrees in writing and editing this manuscript.

FUNDING

This work is supported in part by ARL MAST CTA # W911NF-08-2-0004.

REFERENCES

Blanke, M., Kinnaert, M., Lunze, J., and Staroswiecki, M. (2003). *Diagnosis and Fault-Tolerant Control*. Berlin Heidelberg: Springer-Verlag.

Bruni, F. (2014). Hacking our humanity: sony, security and the end of privacy. *The New York Times*, SR3.

Cam, H., Mouallem, P., Mo, Y., Sinopoli, B., and Nkrumah, B. (2014). "Modeling impact of attacks, recovery, and attackability conditions for situational awareness," in *IEEE Int. Inter-Disciplinary Conf. on Cognitive Methods in situation Awareness and Decision Support* (San Antonio, TX: IEEE), 181–187.

Cardenas, A., Amin, S., and Sastry, S. (2008). "Secure control: towards survivable cyber-physical systems," in *28th Int. Conf. on Distributed Computing Systems Workshops* (Beijing: IEEE), 495–500.

Cassandras, C. G., and Lafortune, S. (2008). *Introduction to Discrete Event Systems*, Vol. 11. New York, NY: Springer.

CPS-FORCES. (2015). Available at: https://www.cps-forces.org/

de la Higuera, C. (2010). *Grammatical Inference: Learning Automata and Grammars*. Cambridge: Cambridge University Press.

Fu, J., Tanner, H. G., and Heinz, J. (2013). "Adaptive planning in unknown environments using grammatical inference," in *Decision and Control (CDC), 2013 IEEE 52nd Annual Conference on* (Florence: IEEE), 5357–5363.

Fu, J., Tanner, H. G., Heinz, J. N., Karydis, K., Chandlee, J., and Koirala, C. (2015). Symbolic planning and control using game theory and grammatical inference. *Eng. Appl. Artif. Intell.* 37, 378–391. doi:10.1016/j.engappai.2014.09.020

García, P., and Ruiz, J. (2004). Learning k-testable and k-piecewise testable languages from positive data. *Grammars* 7, 125–140.

Gold, M. E. (1967). Language identification in the limit. *Inf. Control* 10, 447–474. doi:10.1016/S0019-9958(67)91165-5

Heinz, J., and Rogers, J. (2013). "Learning subregular classes of languages with factored deterministic automata," in *Proceedings of the 13th Meeting on the Mathematics of Language (MoL 13)*, eds A. Kornai and M. Kuhlmann (Sofia, Bulgaria: Association for Computational Linguistics), 64–71.

Hespanha, J., Naghshtabrizi, P., and Xu, Y. (2007). A survey of recent results in networked control systems. *Proc. IEEE* 95, 138–162. doi:10.1109/JPROC.2006.887288

Jordan, J. (2009). When heads roll: assessing the effectiveness of leadership decapitation. *Secur. Stud.* 18, 719–755. doi:10.1080/09636410903369068

Kannappan, P., Karydis, K., Tanner, H. G., Jardine, A., and Heinz, J. (2016). "Incorporating learning modules improves aspects of resilience of supervisory cyber-physical systems," in *24th Mediterranean Conf. on Control and Automation* (Athens: IEEE).

Kendra, J. M., and Wachtendorf, T. (2003). Elements of resilience after the world trade center disaster: reconstitung New York City's emergency operations centre. *Disasters* 27, 37–53. doi:10.1111/1467-7717.00218

Khaitan, S., and McCalley, J. (2014). Design techniques and applications of cyber-physical systems: a survey. *IEEE Syst. J.* 9, 350–365. doi:10.1109/JSYST.2014.2322503

Kim, K.-D., and Kumar, P. (2012). Cyber-physical systems: a perspective at the centennial. *Proc. IEEE* 100, 1287–1308. doi:10.1109/JPROC.2012.2189792

Kundur, D., Feng, X., Mashayekh, S., Liu, S., Zourntos, T., and Butler-Purry, K. (2011). Towards modelling the impact of cyber attacks on a smart grid. *Int. J. Secur. Netw.* 6, 2–13. doi:10.1504/IJSN.2011.039629

Liu, J., Xiao, Y., Li, S., Liang, W., and Chen, C. L. P. (2012). Cyber security and privacy issues in smart grids. *IEEE Commun. Surv. Tutorials* 14, 981–997. doi:10.1109/SURV.2011.122111.00145

Mahmoud, M. S. (2004). *Resilient Control of Uncertain Dynamical Systems*. Berlin Heidelberg: Springer.

Martin, W., White, P., Taylor, F., and Goldberg, A. (2000). "Formal construction of the mathematically analyzed separation kernel," in *Proceedings of the Fifteenth IEEE International Conference on Automated Software Engineering* (Grenoble: IEEE), 133–141.

McNaughton, R., and Papert, S. (1971). *Counter-Free Automata*. Cambridge, MA: MIT Press.

Mo, Y., and Sinopoli, B. (2009). "Secure control against replay attacks," in *47th Annual Allerton Conf. on Communication, Control, and Computing* (Monticello, IL: IEEE), 911–918.

Rieger, C. (2014). "Resilient control systems practical metrics basis for defining mission impact," in *7th Int. Symp. on Resilient Control Systems* (Denver, CO: IEEE), 1–10.

Rieger, C., Gertman, D., and McQueen, M. (2009). "Resilient control systems: next generation design research," in *2nd Conf. on Human System Interactions* (Catania: IEEE), 632–636.

Rushby, J. (1981). Design and verification of secure systems. *ACM SIGOPS Oper. Syst. Rev.* 15, 12–21. doi:10.1145/1067627.806586

Schenato, L., Sinopoli, B., Franceschetti, M., Poolla, K., and Sastry, S. (2007). Foundations of control and estimation over lossy networks. *Proc. IEEE* 95, 163–187. doi:10.1109/JPROC.2006.887306

Tanner, H. G., Jadbabaie, A., and Pappas, G. J. (2007). Flocking in fixed and switching networks. *IEEE Trans. Automat. Contr.* 52, 863–867. doi:10.1109/TAC.2007.895948

Conflict of Interest Statement: The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.