Self-triggered and team-triggered control of networked cyber-physical systems

Cameron Nowzari Jorge Cortés

I. Introduction

This chapter describes triggered control approaches for the coordination of networked cyber-physical systems. Given the coverage of the other chapters of this book, our focus is on self-triggered control and a novel approach we term team-triggered control.

The basic idea behind triggered approaches for controlled dynamical systems is to opportunistically select when to execute certain actions (e.g., update the actuation signal, sense some data, communicate some information) in order to efficiently perform various control tasks. Such approaches trade computation for sensing, actuation, or communication effort and give rise to real-time controllers that do not need to perform these actions continuously, or even periodically, in order for the system to function according to a desired level of performance. Triggered approaches for control become even more relevant in the context of networked cyber-physical systems, where computation, actuation, sensing and communication capabilities might not be collocated. The successful operation of networked systems critically relies on the acquisition and transmission of information across different subsystems, which adds an additional layer of complexity for controller design and analysis. In fact, a given triggered controller might be implementable over some networks and not over others, depending on the requirements imposed by the triggers on the information flow across the individual subsystems.

In event-triggered control, the focus is on detecting events during the system execution that are relevant from the point of view of task completion in order to trigger appropriate actions.

(1) Cameron Nowzari is with the Department of Electrical and Systems Engineering, University of Pennsylvania, Philadelphia, Pennsylvania, USA, cnowzari@seas.upenn.edu

(2) Jorge Cortés is with the Department of Mechanical and Aerospace Engineering, University of California, San Diego, California, USA, cortes@ucsd.edu

Event-triggered strategies result in good performance but require continuous access to some state or output to be able to monitor the possibility of an event being triggered at any time. This can be especially costly to implement when executed over networked systems (for instance, if the availability of information relies on continuous agent-to-agent communication). Other chapters in this book alleviate this issue by considering periodic or data-sampled event-triggered control, where the triggers are only checked when information is available, but this may still be inefficient if the events are not triggered often.

In self-triggered control, the emphasis is instead on developing tests that rely only on current information available to the decision maker to schedule future actions (e.g., when to measure the state again, when to recompute the control input and update the actuator signals, or when to communicate information). To do so, this approach relies critically on abstractions about how the system might behave in the future given the information available now. Self-triggered strategies can be made robust to uncertainties (since they can naturally be accounted for in the abstractions) and more easily amenable to distributed implementation. However, they often result in conservative executions because of the inherent over-approximation about the state of the system associated to the given abstractions. Intuitively, it makes sense to believe that the more accurate a given abstraction describes the system behavior, the less conservative the resulting self-triggered controller implementation. This is especially so in the context of networked systems, where individual agents need to maintain various abstractions about the state of their neighbors, the environment, or even the network.

The team-triggered approach builds on the strengths of event- and self-triggered control to synthesize a unified approach for controlling networked systems in real time that combines the best of both worlds. The approach is based on agents making promises to one another about their future states and being responsible for warning each other if they later decide to break them. This is reminiscent of event-triggered implementations. Promises can be broad, from tight state trajectories to loose descriptions of reachability sets. With the information provided by promises, individual agents can autonomously determine what time in the future fresh information is needed to maintain a desired level of performance. This is reminiscent of self-triggered implementations. A benefit of the strategy is that because of the availability of the promises, agents do not require continuous state information about their neighbors. Also, because of the extra information provided by promises about what other agents plan to do, agents can operate more efficiently

and less conservatively compared to self-triggered strategies where worst-case conditions must always be considered.

A. Related work

The other chapters of this book provide a broad panoramic view of the relevant literature on general event- and self-triggered control of dynamical systems [1], [2], [3], so we focus here on networked systems understood in a broad sense. The predominant paradigm is that of a single plant that is stabilized through a decentralized triggered controller over a sensor-actuator network, see e.g. [4], [5], [6]. Fewer works have considered scenarios where multiple plants or agents together are the subject of the overall control design, where the synthesis of appropriate triggers raises novel challenges due to the lack of centralized decision making and the local agent-to-agent interactions. Exceptions include consensus via event-triggered [7], [8], [9] or self-triggered control [7], rendezvous [10], collision avoidance while performing point-to-point reconfiguration [11], distributed optimization [12], [13], [14], model predictive control [15], and model-based event-triggered control [16], [17]. The works in [7], [18] implement self-triggered communication schemes to perform distributed control where agents assume worst-case conditions for other agents when deciding when new information should be obtained. Distributed strategies based on event-triggered communication and control are explored in [19], where each agent has an a priori determined local error tolerance, and once it violates it the agent broadcasts its updated state to its neighbors. The same event-triggered approach is taken in [20] to implement gradient control laws that achieve distributed optimization. In the interconnected system considered in [16], each subsystem helps neighboring subsystems by monitoring their estimates and ensuring that they stay within some performance bounds. The approach requires different subsystems to have synchronized estimates of one another even though they do not communicate at all times. In [21], [22], agents do not have continuous availability of information from neighbors and instead decide when to broadcast new information to them, which is more in line with the work we present here.

B. Notation

Here we collect some basic notation used throughout the paper. We let $\mathbb{R}$, $\mathbb{R}_{\geq 0}$, and $\mathbb{Z}_{\geq 0}$ be the sets of real, nonnegative real, and nonnegative integer numbers, respectively. We use $\|\cdot\|$ to

denote the Euclidean distance. Given p and $q \in \mathbb{R}^d$, $[p, q] \subset \mathbb{R}^d$ is the closed line segment with extreme points p and q, and $\overline{B}(p, r) = \{q \in \mathbb{R}^d \mid \|q - p\| \leq r\}$ is the closed ball centered at p with radius $r \in \mathbb{R}_{\geq 0}$. Given $v \in \mathbb{R}^d \setminus \{0\}$, $\mathrm{unit}(v)$ is the unit vector in the direction of v. A function $\alpha : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ belongs to class $\mathcal{K}$ if it is continuous, strictly increasing, and $\alpha(0) = 0$. A function $\beta : \mathbb{R}_{\geq 0} \times \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ belongs to class $\mathcal{KL}$ if for each fixed r, $\beta(r, s)$ is nonincreasing with respect to s and $\lim_{s \to \infty} \beta(r, s) = 0$, and for each fixed s, $\beta(\cdot, s) \in \mathcal{K}$. For a set S, we let ∂S denote its boundary and $\mathbb{P}^{\mathrm{cc}}(S)$ denote the collection of compact and connected subsets of S.

II. Self-triggered control of a single plant

In this section, we review the main ideas behind the self-triggered approach for the control of a single plant. Our starting point is the availability of a controller that, when implemented in continuous time, asymptotically stabilizes the system and is certified by a Lyapunov function. For a constant input, the decision maker can compute the future evolution of the system given the current state (using abstractions of the system behavior), and determine what time in the future the controller should be updated. For simplicity, we present the discussion for a linear control system, although the results are valid for more general scenarios, such as nonlinear systems for which a controller is available that makes the closed-loop system input-to-state stable. The exposition closely follows [23].

A. Problem statement

Consider a linear control system

$$\dot{x} = Ax + Bu, \tag{1}$$

with $x \in \mathbb{R}^n$ and $u \in \mathbb{R}^m$, for which there exists a linear feedback controller $u = Kx$ such that the closed-loop system

$$\dot{x} = (A + BK)x,$$

is asymptotically stable. Given a positive definite matrix $Q \in \mathbb{R}^{n \times n}$, let $P \in \mathbb{R}^{n \times n}$ be the unique solution to the Lyapunov equation $(A + BK)^T P + P(A + BK) = -Q$. Then, the evolution of the Lyapunov function $V_c(x) = x^T P x$ along the trajectories of the closed-loop system is

$$\dot{V}_c = x^T((A + BK)^T P + P(A + BK))x = -x^T Q x.$$

Consider now a sample-and-hold implementation of the controller, where the input is not updated continuously, but instead at a sequence of to-be-determined times $\{t_\ell\}_{\ell\in\mathbb{Z}_{\geq 0}} \subset \mathbb{R}_{\geq 0}$,

$$u(t) = Kx(t_\ell), \quad t \in [t_\ell, t_{\ell+1}). \tag{2}$$

Such an implementation makes sense in practical scenarios given the inherent nature of digital systems. Under this controller implementation, the closed-loop system can be written as

$$\dot{x} = (A + BK)x + BKe,$$

where $e(t) = x(t_\ell) - x(t)$, $t \in [t_\ell, t_{\ell+1})$, is the state error. Then, the objective is to determine the sequence of times $\{t_\ell\}_{\ell\in\mathbb{Z}_{\geq 0}}$ to guarantee some desired level of performance for the resulting system. To make this concrete, define the function

$$V(t, x_0) = x(t)^T P x(t),$$

for a given initial condition $x(0) = x_0$ (here, $t \mapsto x(t)$ denotes the evolution of the closed-loop system using (2)). We define the performance of the system by means of a function $S : \mathbb{R}_{\geq 0} \times \mathbb{R}^n \to \mathbb{R}_{\geq 0}$ that upper bounds the evolution of V. Then, the sequence of times $\{t_\ell\}$ can be implicitly defined as the times at which

$$V(t, x_0) \leq S(t, x_0) \tag{3}$$

is not satisfied. As stated, this is an event-triggered condition, that updates the actuator signal whenever $V(t_\ell, x_0) = S(t_\ell, x_0)$. Assuming solutions are well defined, it is not difficult to see that if the performance function satisfies $S(t, x_0) \leq \beta(t, |x_0|)$, for some $\beta \in \mathcal{KL}$, then the closed-loop system is globally uniformly asymptotically stable. Moreover, if β is an exponential function, the system is globally uniformly exponentially stable.

Therefore, one only needs to guarantee the lack of Zeno behavior. We do this by choosing the performance function S so that the inter-event times $t_{\ell+1} - t_\ell$ are lower bounded by some constant positive quantity. This can be done in a number of ways. For the linear system (1), it turns out that it is sufficient to select S satisfying $\dot{V}(t_\ell) < \dot{S}(t_\ell)$ at the event times t_ℓ (this fact is formally stated below in Theorem II.1). To do so, choose $R \in \mathbb{R}^{n\times n}$ positive definite such that $Q - R$ is also positive definite. Then, there exists a Hurwitz matrix $A_s \in \mathbb{R}^{n\times n}$ such that the Lyapunov equation

$$A_s^T P + P A_s = -R$$

holds. Consider the hybrid system,

$$\dot{x}_s = A_s x_s, \quad t \in [t_\ell, t_{\ell+1}),$$
$$x_s(t_\ell) = x(t_\ell),$$

whose trajectories we denote by $t \mapsto x_s(t)$, and define the performance function S by

$$S(t) = x_s^T(t) P x_s(t).$$

B. Self-triggered control policy

Under the event-triggered controller implementation described above, the decision maker needs access to the exact state at all times in order to monitor the condition (3). Here we illustrate how the self-triggered approach, instead, proceeds to construct an abstraction that captures the system behavior given the available information and uses it to determine what time in the future the condition might be violated (eliminating the need to continuously monitor the state).

For the particular case considered here, where the dynamics (1) are linear and deterministic, and no disturbances are present, it is a possible to exactly compute, at any given time t_ℓ, the future evolution of the state, and hence how long $t_{\ell+1} - t_\ell$ will elapse until the condition (3) is enabled again. In general this is not possible, and one instead builds an abstraction that over-approximates this evolution, resulting in a trigger that generates inter-event times which lower bound the inter-event times generated by (3). This explains why the self-triggered approach generally results in more conservative implementations than the event-triggered approach, but comes with the benefit of not requiring continuous state information.

Letting $y = [x^T, e^T]^T \in \mathbb{R}^n \times \mathbb{R}^n$, we write the continuous-time dynamics as

$$\dot{y} = Fy, \quad t \in [t_\ell, t_{\ell+1}),$$

where

$$F = \begin{bmatrix} A + BK & BK \\ -A - BK & -BK \end{bmatrix}.$$

With a slight abuse of notation, we let $y_\ell = [x^T(t_\ell), 0^T]^T$ be the state y at time t_ℓ. Note that $e(t_\ell) = 0$, for all $\ell \in \mathbb{Z}_{\geq 0}$, by definition of the update times. With this notation, we can rewrite

$$S(t) = (Ce^{F_s(t-t_\ell)} y_\ell)^T P (Ce^{F_s(t-t_\ell)} y_\ell),$$
$$V(t) = (Ce^{F(t-t_\ell)} y_\ell)^T P (Ce^{F(t-t_\ell)} y_\ell),$$

where

$$F_s = \begin{bmatrix} A_s & 0 \\ 0 & 0 \end{bmatrix}, \quad C = \begin{bmatrix} I & 0 \end{bmatrix}.$$

The condition (3) can then be rewritten as

$$f(t, y_\ell) = y_\ell^T (e^{F^T(t-t_\ell)} C^T P C e^{F(t-t_\ell)} - e^{F_s^T(t-t_\ell)} C^T P C e^{F_s(t-t_\ell)}) y_\ell \leq 0.$$

What is interesting about this expression is that it clearly reveals the important fact that, with the information available at time t_ℓ, the decision maker can determine the next time $t_{\ell+1}$ at which (3) is violated by computing $t_{\ell+1} = h(x(t_\ell))$ as the time for which

$$f(h(x(t_\ell)), y_\ell) = 0. \tag{4}$$

The following result from [23] provides a uniform lower bound $t_{\min}$ on the inter-event times $\{t_{\ell+1} - t_\ell\}_{\ell \in \mathbb{Z}_{\geq 0}}$.

Theorem II.1 (Lower bound on inter-event times for self-triggered approach) *Given the system* (1) *with controller* (2) *and controller updates given by the self-triggered policy* (4)*, the inter-event times are lower bounded by*

$$t_{\min} = \min\{t \in \mathbb{R}_{>0} \mid \det(M(t)) = 0\} > 0,$$

where

$$M(t) = \begin{bmatrix} I & 0 \end{bmatrix} \left(e^{Ft} C^T P C e^{Ft} - e^{F_s t} C^T P C e^{F_s t}\right) \begin{bmatrix} I \\ 0 \end{bmatrix}.$$

Note that the above result can also be interpreted in the context of a periodic controller implementation: any period less than or equal to $t_{\min}$ results in a closed-loop system with asymptotic stability guarantees.

Remark II.2 (Triggered sampling vs triggered control) We must make an important distinction here between triggered *control* and triggered *sampling* (or more generally, any type of data acquisition). Due to the deterministic nature of the general linear system discussed in this section, the answers to the questions

- *when should a sample of the state be taken?*
- *when should the control signal to the system be updated?*

are identical. In general, if there are disturbances in the system dynamics, the answers to these questions will not coincide. This might be further complicated in cases when the sensor and the actuator are not collocated, and the sharing of information between them is done via communication. The simplest implementation in this case would be to update the control signal each time a sample is taken; however, in cases where computing or applying a control signal is expensive, it may be more beneficial to use a hybrid strategy combining self-triggered sampling with event-triggered control, as in [24]. •

III. Distributed self-triggered control

In this section, we expand our previous discussion with a focus on networked cyber-physical systems. Our main objective is to explain and deal with the issues that come forward when pushing the triggered approach from controlling a single plant, as considered in Section II, to coordinating multiple cooperating agents. We consider networked systems whose interaction topology is modeled by a graph that captures how information is transmitted (via, for instance, sensing or communication). In such scenarios, each individual agent does not have access to all the network information at any given time, but instead interacts with a set of neighboring agents. In our forthcoming discussion, we pay attention to the following challenges specific to these networked problems. Firstly, the trigger designs now need to be made for individuals, as opposed to a single centralized decision maker. Triggers such as (3) or (4) rely on global information about the network state, which the agents do not possess in general. Therefore, the challenge is finding ways in which such conditions can be allocated among the individuals. This brings to the forefront the issue of access to information and the need for useful abstractions about the behavior of other agents. Secondly, the design of independent triggers that each agent can evaluate with the information available to it naturally gives rise to asynchronous network executions, in which events are triggered by different agents at different times. This 'active' form of asynchronism poses challenges for the correctness and convergence analysis of the coordination algorithms, and further complicates ruling out the presence of Zeno behavior. Before getting into the description of triggered control approaches, we provide a formal description of the model for networked cyber-physical systems and the overall team objective.

A. Network model and problem statement

We begin by describing the model for the network of agents. In order to simplify the general discussion, we make a number of simplifying assumptions, such as linear agent dynamics or time-invariant interaction topology. However, when considering more specific classes of problems, it is often possible to drop some of these assumptions.

Consider N agents whose interaction topology is described by an undirected graph $\mathcal{G}$. The fact that (i, j) belongs to the edge set E models the ability of agents i and j to communicate with one another. The set of all agents that i can communicate with is given by its set of neighbors $\mathcal{N}(i)$ in the graph $\mathcal{G}$. The state of agent $i \in \{1, \dots, N\}$, denoted x_i, belongs to a closed set $\mathcal{X}_i \subset \mathbb{R}^{n_i}$. We assume that each agent has access to its own state at all times. According to our model, agent i can access $x^i_{\mathcal{N}} = \big(x_i, \{x_j\}_{j\in\mathcal{N}(i)}\big)$ when it communicates with its neighbors. The network state $x = (x_1, \dots, x_N)$ belongs to $\mathcal{X} = \prod_{i=1}^N \mathcal{X}_i$. We consider linear dynamics for each $i \in \{1, \dots, N\}$,

$$\dot{x}_i = A_i x_i + B_i u_i, \tag{5}$$

with block diagonal $A_i \in \mathbb{R}^{n_i\times n_i}$, $B_i \in \mathbb{R}^{n_i\times m_i}$, and $u_i \in \mathcal{U}_i$. Here, $\mathcal{U}_i \subset \mathbb{R}^{m_i}$ is a closed set of allowable controls for agent i. We assume that the pair (A_i, B_i) is controllable with controls taking values in $\mathcal{U}_i$. Letting $u = (u_1, \dots, u_N) \in \prod_{i=1}^N \mathcal{U}_i$, the dynamics of the entire network is described by

$$\dot{x} = Ax + Bu, \tag{6}$$

with $A = \operatorname{diag}(A_1, \dots, A_N) \in \mathbb{R}^{n\times n}$ and $B = \operatorname{diag}(B_1, \dots, B_N) \in \mathbb{R}^{n\times m}$, where $n = \sum_{i=1}^N n_i$, and $m = \sum_{i=1}^N m_i$.

The goal of the network is to drive the agents' states to some desired closed set of configurations $D \subset \mathcal{X}$ and ensure that it stays there. Depending on how the set D is defined, this goal can capture different coordination tasks, including deployment, rendezvous, and formation control, see e.g. [25]. Our objective here is to design provably correct triggered coordination strategies that agents can implement to achieve this goal. With this in mind, given the agent dynamics, the communication graph $\mathcal{G}$, and the set D, our starting point is the availability of a distributed, continuous control law that drives the system asymptotically to D. Formally, we assume that a continuous map $u^* : \mathcal{X} \to \mathbb{R}^m$ and a continuously differentiable function $V : \mathcal{X} \to \mathbb{R}$, bounded

from below exist such that D is the set of minimizers of V and, for all $x \notin D$,

$$\nabla_i V(x)\left(A_i x_i + B_i u_i^*(x)\right) \leq 0, \quad i \in \{1, \ldots, N\}, \tag{7a}$$

$$\sum_{i=1}^{N} \nabla_i V(x)\left(A_i x_i + B_i u_i^*(x)\right) < 0. \tag{7b}$$

We assume that both the control law u^* and the gradient ∇V are distributed over $\mathcal{G}$. By this we mean that, for each $i \in \{1, \ldots, N\}$, the ith component of each of these objects only depends on $x^i_{\mathcal{N}}$, rather than on the full network state x. For simplicity, and with a slight abuse of notation, we write $u_i^*(x^i_{\mathcal{N}})$ and $\nabla_i V(x^i_{\mathcal{N}})$ to emphasize this fact when convenient. This property has the important consequence that agent i can compute u_i^* and $\nabla_i V$ with the exact information it can obtain through communication on $\mathcal{G}$. We note that here we only consider the goal of achieving asymptotic stability of the system, with no specification on the performance S as we did in Section II. For specific problems, one could reason with similar performance function requirements.

From an implementation viewpoint, the controller u^* requires continuous agent-to-agent communication and continuous updates of the actuator signals. The triggered coordination strategies that we review next seek to select the time instants at which information should be acquired in an opportunistic way, such that the resulting implementation still enjoys certificates on correctness and performance similar to that of the original controller. The basic general idea is to design implementations that guarantee that the time derivative of the Lyapunov function V introduced in Section III-A along the solutions of the network dynamics (6) is less than or equal to 0 at all times, even though the information available to the agents is partial and possibly inexact.

Remark III.1 (Triggered sampling, communication, and control) In light of Remark II.2, the consideration of a networked cyber-physical system adds a third independent question that each agent of the network must answer:

- *when should I sample the (local) state of the system?*
- *when should I update the (local) control input to the system?*
- *when should I communicate with my neighbors?*

Even more, to answer these questions, the decision maker has now changed from a single one with complete information on a single plant to multiple agents with partial information about the network. For simplicity, and to emphasize the network aspect, we assume that agents have

continuous access to their own state and that control signals are updated each time communication with neighbors occur. We refer the interested reader to [16], [20] where these questions are considered independently in specific classes of systems. •

B. Event-triggered communication and control

In this section we see how an event-triggered implementation of the continuous control law u^* might be realized using (7).

Let t_ℓ for some $\ell \in \mathbb{Z}_{\geq 0}$ be the last time at which all agents have received information from their neighbors. We are then interested at what time $t_{\ell+1}$ the agents should communicate again. In-between updates, the simplest estimate that an agent i maintains about a neighbor $j \in \mathcal{N}(i)$ is the zero-order hold given by

$$\widehat{x}_j^i(t) = x_j(t_\ell), \qquad t \in [t_\ell, t_{\ell+1}). \tag{8}$$

Recall that agent i always has access to its own state. Therefore, $\widehat{x}_{\mathcal{N}}^i(t) = (x_i(t), \{\widehat{x}_j^i(t)\}_{j\in\mathcal{N}(i)})$ is the information available to agent i at time t. The implemented controller is then given by

$$\widehat{x}_j^i(t) = x_j(t_\ell), \qquad u_i^{\text{event}}(t) = u_i^*(\widehat{x}_{\mathcal{N}}^i(t_\ell)),$$

for $t \in [t_\ell, t_{\ell+1})$. We only consider the zero-order hold here for simplicity, although one could also implement more elaborate schemes for estimating the state of neighboring agents given the sampled information available, and use this for the controller implementation.

The time $t_{\ell+1}$ at which an update becomes necessary is determined by the first time after t_ℓ that the time derivative of V along the trajectory of (6) with $u = u^{\text{event}}$ is no longer negative. Formally, the event for when agents should request updated information is

$$\frac{d}{dt}V(x(t_{\ell+1})) = \sum_{i=1}^{N} \nabla_i V(x(t_{\ell+1}))\left(A_i x_i(t_{\ell+1}) + B_i u_i^{\text{event}}(t_\ell)\right) = 0. \tag{9}$$

Two reasons may cause (9) to be satisfied. One reason is that the zero-order hold control $u^{\text{event}}(t)$ for $t \in [t_\ell, t_{\ell+1})$ has become too outdated, causing an update at time $t_{\ell+1}$. Until (9) is satisfied, it is not necessary to update state information because inequality (7b) implies that $\frac{d}{dt}V(x(t)) < 0$ for $t \in [t_\ell, t_{\ell+1})$ if $x(t_\ell) \notin D$ given that all agents have exact information at t_ℓ and $\frac{d}{dt}V(x(t_\ell)) < 0$ by continuity of $\frac{d}{dt}V(x)$. The other reason is simply that $x(t_{\ell+1}) \in D$ and thus the system has reached its desired configuration.

Unfortunately, (9) cannot be checked in a distributed way because it requires global information. Instead, one can define a local event that implicitly defines when a single agent $i \in \{1, \dots, N\}$ should update its information. Letting t_ℓ^i be some time at which agent i receives updated information, $t_{\ell+1}^i \geq t_\ell^i$ is the first time such that

$$\nabla_i V(x(t_{\ell+1}^i)) \left(A_i x_i(t_{\ell+1}^i) + B_i u_i^{\text{event}}(t_\ell^i) \right) = 0. \tag{10}$$

This means that as long as each agent i can ensure the local event (10) has not yet occurred, it is guaranteed that (9) has not yet occurred either. Note that this is a sufficient condition for (9) not being satisfied, but not necessary. There may be other ways of distributing the global trigger (9) for more specific classes of problems. Although this can now be monitored in a distributed way, the problem with this approach is that each agent $i \in \{1, \dots, N\}$ needs to have continuous access to information about the state of its neighbors $\mathcal{N}(i)$ in order to evaluate $\nabla_i V(x) = \nabla_i V(x^i_{\mathcal{N}})$ and check condition (10). This requirement may make the event-triggered approach impractical when this information is only available through communication. Note that this does not mean it is impossible to perform event-triggered communication strategies in a distributed way, but it requires a way for agents to use only their own information to monitor some meaningful event. An example problem where this can be done is provided in [26], [27], where agents use event-triggered broadcasting to solve the consensus problem.

C. Self-triggered communication and control

In this section we design a self-triggered communication and control strategy of the controller u^* by building on the discussion in Section III-B. To achieve this, the basic idea is to remove the requirement on continuous availability of information to check the test (10) by providing agents with possibly inexact information about the state of their neighbors.

To do so, we begin by introducing the notion of reachability sets. Given $y \in \mathcal{X}_i$, let $\mathcal{R}_i(s, y)$ be the set of reachable points under (5) starting from y in s seconds,

$$\mathcal{R}_i(s, y) = \{z \in \mathcal{X}_i \mid \exists\, u_i : [0, s] \to \mathcal{U}_i \text{ such that } z = e^{A_i s} y + \int_0^s e^{A_i(s-\tau)} B_i u_i(\tau) d\tau\}.$$

Assuming here that agents have exact knowledge about the dynamics and control sets of their neighboring agents, each time an agent receives state information, it can construct sets that are

guaranteed to contain their neighbors' states in the future. Formally, if t_ℓ is the time at which agent i receives state information $x_j(t_\ell)$ from its neighbor $j \in \mathcal{N}(i)$, then

$$\mathbf{X}_j^i(t, x_j(t_\ell)) = \mathcal{R}_j(t - t_\ell, x_j(t_\ell)) \subset \mathcal{X}_j \tag{11}$$

is guaranteed to contain $x_j(t)$ for all $t \geq t_\ell$. We refer to these sets as *guaranteed sets*. These capture notion of abstractions of the system behavior that we discussed in Section II when synthesizing a self-triggered control policy for a single plant. For simplicity, we let $\mathbf{X}_j^i(t) = \mathbf{X}_j^i(t, x_j(t_\ell))$ when the starting state $x_j(t_\ell)$ and time t_ℓ do not need to be emphasized. We denote by $\mathbf{X}_{\mathcal{N}}^i(t) = (x_i(t), \{\mathbf{X}_j^i(t)\}_{j \in \mathcal{N}(i)})$ the information available to an agent i at time t.

With the guaranteed sets in place, we can now provide a test using inexact information to determine when new, up-to-date information is required. Let t_ℓ^i be the last time at which agent i received updated information from its neighbors. Until the next time $t_{\ell+1}^i$ information is obtained, agent i uses the zero-order hold estimate and control

$$\widehat{x}_j^i(t) = x_j(t_\ell^i), \qquad u_i^{\text{self}}(t) = u_i^*(\widehat{x}_{\mathcal{N}}^i(t_\ell^i)),$$

for $t \in [t_\ell^i, t_{\ell+1}^i)$ and $j \in \mathcal{N}(i)$. As noted before, the consideration of zero-order hold is only made to simplify the exposition. At time t_ℓ^i, agent i computes the next time $t_{\ell+1}^i \geq t_\ell^i$ at which information should be acquired via

$$\sup_{y_{\mathcal{N}} \in \mathbf{X}_{\mathcal{N}}^i(t_{\ell+1}^i)} \nabla_i V(y_{\mathcal{N}}) \left(A_i x_i(t_{\ell+1}^i) + B_i u_i^{\text{self}}(t_\ell^i)\right) = 0. \tag{12}$$

By (7a) and the fact that $\mathbf{X}_j^i(t_\ell^i) = \{x_j(t_\ell^i)\}$, at time t_ℓ^i we have

$$\sup_{y_{\mathcal{N}} \in \mathbf{X}_{\mathcal{N}}^i(t_\ell^i)} \nabla_i V(y_{\mathcal{N}}) \left(A_i x_i(t_\ell^i) + B_i u_i^{\text{self}}(t_\ell^i)\right) = \nabla_i V(x_{\mathcal{N}}^i(t_\ell^i)) \left(A_i x_i(t_\ell^i) + B_i u_i^{\text{self}}(t_\ell^i)\right) \leq 0.$$

If all agents use this triggering criterium for updating information, it is guaranteed that $\frac{d}{dt}V(x(t)) \leq 0$ at all times because, for each $i \in \{1, \ldots, N\}$, the true state $x_j(t)$ is guaranteed to be in $\mathbf{X}_j^i(t)$ for all $j \in \mathcal{N}(i)$ and $t \geq t_\ell^i$.

The condition (12) is appealing because it can be solved by agent i with the information it possesses at time t_ℓ^i. Once determined, agent i schedules that, at time $t_{\ell+1}^i$, it will request updated information from its neighbors. The term self-triggered captures the fact that each agent is now responsible for deciding when it requires new information. We refer to $t_{\ell+1}^i - t_\ell^i$ as the *self-triggered request time* of agent $i \in \{1, \ldots, N\}$. Due to the conservative way in which $t_{\ell+1}^i$ is

determined, it is possible that $t^i_{\ell+1} = t^i_\ell$ for some i, which would mean that continuous information updates are necessary (it should be noted that this cannot happen for all $i \in \{1, \dots, N\}$ unless the network state is already in D). For a given specific task (target set D, Lyapunov function V, and controller u^*), one might be able to discard this issue. In other cases, this can be dealt with by introducing a dwell time such that a minimum amount of time must pass before an agent can request new information. We do not enter into the details of this here, but instead show how it can be done for a specific example in Section V.

The main convergence result for the distributed self-triggered approach is provided below. We defer the sketch of the proof to Section IV as it becomes a direct consequence of Proposition IV.2.

Proposition III.2 *For the network model and problem setup described in Section III-A, if each agent executes the self-triggered algorithm presented above, the state of the entire network asymptotically converges to the desired set D for all Zeno-free executions.*

Note that the result is formulated for non-Zeno executions. As discussed above, dwell times can be used to rule out Zeno behavior although we do not go into details here. The problem with the distributed self-triggered approach is that the resulting times are often conservative because the guaranteed sets can grow large quickly as they capture all possible trajectories of neighboring agents. It is conceivable that improvements can be made from tuning the guaranteed sets based on what neighboring agents *plan* to do rather than what they *can* do. This observation is at the core of the team-triggered approach proposed next.

IV. TEAM-TRIGGERED CONTROL

Here we describe an approach that combines ideas from both event- and self-triggered control for the coordination of networked cyber-physical systems. The team-triggered strategy incorporates the reactive nature of event-triggered approaches and, at the same time, endows individual agents with the autonomy characteristics of self-triggered approaches to determine when and what information is needed. Agents make promises to their neighbors about their future states and inform them if these promises are violated later (hence the connection with event-triggered control). These promises function as more accurate abstractions of the behavior of the other entities than guaranteed sets introduced in Section III-C. This extra information allows each agent to compute the next time that an update is required (which in general is longer than in

the purely self-triggered implementation described in Section III) and request information from their neighbors to guarantee the monotonicity of the Lyapunov function (hence the connection with self-triggered control). Our exposition here follows [28].

A. Promise sets

Promises allow agents to predict the evolution of their neighbors more accurately than guaranteed sets, which in turn affects the overall network behavior. For our purposes here, a (control) *promise* that agent j makes to agent i at some time t' is a subset $\mathcal{U}_j^i \subset \mathcal{U}_j$ of its allowable control set. This promise conveys the meaning that agent j will only use controls $u(t) \in \mathcal{U}_j^i$ for all $t \geq t'$. Given the dynamics (5) of agent j and state $x_j(t')$ at time t', agent i can compute the *state promise set*,

$$X_j^i(t) = \{z \in \mathcal{X}_j \mid \exists\, u_j : [t', t] \to \mathcal{U}_j^i \text{ s.t. } z = e^{A_j(t-t')}x_j(t) + \int_{t'}^{t} e^{A_j(t-\tau)}B_j u_j(\tau)d\tau\}, \quad (13)$$

for $t \geq t'$. This means that as long as agent j keeps its promise at all times (i.e., $u_j(t) \in \mathcal{U}_j^i$ for all $t \geq t'$), then $x_j(t) \in X_j^i(t)$ for all $t \geq t'$. We denote by $X_{\mathcal{N}}^i(t) = (x_i(t), \{X_j^i(t)\}_{j\in\mathcal{N}(i)})$ the information available to an agent i at time t. Since the promise $\mathcal{U}_j^i$ is a subset of agent j's allowable control set $\mathcal{U}_j$, it follows that $X_j^i(t) \subset \mathbf{X}_j^i(t)$ for all $t \geq t'$ as well. This allows agent i to operate with better information about agent j than it would by computing the guaranteed sets (11) defined in Section III-C. Promises can be generated in different ways depending on the desired task. For simplicity, we only consider static promises $\mathcal{U}_j^i$ here, although one could also consider more complex time-varying promises [28].

In general, tight promises correspond to agents having good information about their neighbors, which at the same time may result in an increased communication effort (since the promises cannot be kept for long periods of time). Loose promises correspond to agents having to use more conservative controls due to the lack of information, while at the same time potentially being able to operate for longer periods of time without communicating (because promises are not violated). These advantages rely on the assumption that promises hold throughout the evolution. As the state of the network changes and the level of task completion evolves, agents might decide to break former promises and make new ones. We discuss this process in Section IV-C.

Remark IV.1 (Controllers that operate on set-valued information) With the extra information provided by promise sets, it is conceivable to design controllers that operate on set-valued

information rather than point-valued information and go beyond zero-order state holds. For simplicity, we only consider controllers that operate on points in our general exposition, but illustrate how set-valued controllers can be defined in Section V. We refer the interested reader to [28] for further details. •

B. Self-triggered communication and control

With the promise sets in place, we can now provide a test using the available information to determine when new, up-to-date information is required.

Let t_ℓ^i be the last time at which agent i received updated information from its neighbors. Until the next time $t_{\ell+1}^i$ information is obtained, agent i uses the zero-order hold estimate and control

$$\widehat{x}_j^i(t) = x_j(t_\ell^i), \qquad u_i^{\text{team}}(t) = u_i^*(\widehat{x}_{\mathcal{N}}^i(t_\ell^i)),$$

for $t \in [t_\ell^i, t_{\ell+1}^i)$ and $j \in \mathcal{N}(i)$. At time t_ℓ^i, agent i computes the next time $t_{\ell+1}^i \geq t_\ell^i$ at which information should be acquired via

$$\sup_{y_{\mathcal{N}} \in X_{\mathcal{N}}^i(t_{\ell+1}^i)} \nabla_i V(y_{\mathcal{N}}) \left(A_i x_i(t_{\ell+1}^i) + B_i u_i^{\text{team}}(t_\ell^i)\right) = 0. \tag{14}$$

By (7a) and the fact that $X_j^i(t_\ell^i) = \{x_j(t_\ell^i)\}$, at time t_ℓ^i we have

$$\sup_{y_{\mathcal{N}} \in \mathbf{X}_{\mathcal{N}}^i(t_\ell^i)} \nabla_i V(y_{\mathcal{N}}) \left(A_i x_i(t_\ell^i) + B_i u_i^{\text{team}}(t_\ell^i)\right) = \nabla_i V(x_{\mathcal{N}}^i(t_\ell^i)) \left(A_i x_i(t_\ell^i) + B_i u_i^{\text{team}}(t_\ell^i)\right) \leq 0.$$

If all agents use this triggering criterium for updating information, and all promises among agents are kept at all times, it is guaranteed that $\frac{d}{dt}V(x(t)) \leq 0$ at all times because, for each $i \in \{1, \ldots, N\}$, the true state $x_j(t)$ is guaranteed to be in $X_j^i(t)$ for all $j \in \mathcal{N}(i)$ and $t \geq t_\ell^i$.

As long as (14) has not yet occurred for all agents $i \in \{1, \ldots, N\}$ for some time t and the promises have not been broken, assumptions (7) and the continuity of the LHS of (14) in time guarantee

$$\frac{d}{dt}V(x(t)) \leq \sum_{i=1}^{N} \mathcal{L}_i V^{\text{sup}}(X_{\mathcal{N}}^i(t)) < 0. \tag{15}$$

The condition (14) is again appealing because it can be solved by agent i with the information it possesses at time t_ℓ^i. In general, it gives rise to a longer inter-event time than the condition (12), because the agents employ more accurate information about the future behavior of their neighbors provided by the promises. However, as in the self-triggered method described in Section III-C

and depending on the specific task, it is possible that $t^i_{\ell+1} = t_\ell$ for some agent i, which means this agent would need continuous information from its neighbors. If this is the case, the issue can be dealt with by introducing a dwell time such that a minimum amount of time must pass before an agent can request new information. We do not enter into the details of this here, but instead refer to [28] for a detailed discussion. Finally, we note that the conclusions laid out above only hold if promises among agents are kept at all times, which we address next.

C. Event-triggered information updates

Agent promises may need to be broken for a variety of reasons. For instance, an agent might receive new information from its neighbors, causing it to change its former plans. Another example is given by an agent that made a promise that is not able to keep for as long as it anticipated. Consider an agent $i \in \{1, \dots, N\}$ that has sent a promise $\mathcal{U}_i^j$ to a neighboring agent j at some time t_{last}, resulting in the state promise set $x_i(t) \in X_i^j(t)$. If agent i ends up breaking its promise at time $t^* > t_{\text{last}}$, then it is responsible for sending a new promise $\mathcal{U}_i^j$ to agent j at time $t_{\text{next}} = t^*$. This implies that agent i must keep track of promises made to its neighbors and monitor them in case they are broken. This mechanism is implementable because each agent only needs information about its own state and the promises it has made to determine whether the trigger is satisfied. We also note that different notions of 'broken promise' are acceptable: this might mean that $u_i(t^*) \notin \mathcal{U}_i^j$ or the more flexible $x_i(t^*) \notin X_i^j(t^*)$. The latter has an advantage in that it is not absolutely required that agent i use controls only in $\mathcal{U}_i^j$, as long as the promise set that agent j is operating with is still valid.

Reminiscent of the discussion on dwell times in Section IV-B, it may be possible for promises to broken arbitrarily fast, resulting in Zeno executions. As before, it is possible to implement a dwell time such that a minimum amount of time must pass before an agent can generate a new promise. However, in this case, some additional consideration must be given to the fact that agents might not always be operating with correct information in this case. For instance, if an agent j has broken a promise to agent i but is not allowed to send a new one yet due to the dwell time, agent i might be operating with false information. Such a problem can be addressed using small warning messages that alert agents of this situation. Another issue that is important when considering the transmission of messages is that communication might be unreliable, with delays, packet drops, or noise. We refer to [28] for details on how these dwell times can be implemented

while ensuring stability of the entire system in the presence of unreliable communication.

D. Convergence analysis of the team-triggered coordination policy

The combination of the self-triggered communication and control policy with the event-triggered information updates gives rise to the team-triggered coordination policy, which is formally presented in Algorithm 1. The self-triggered information request in this strategy is executed by an agent anytime new information is received, whether it was actively requested by the agent, or was received from some neighbor due to the breaking of a promise.

Algorithm 1: Team-triggered coordination policy

(Self-triggered communication and control)
At any time t agent $i \in \{1, \dots, N\}$ receives new promise(s) $\mathcal{U}_j^i$ and position information $x_j(t)$ from neighbor(s) $j \in \mathcal{N}(i)$, agent i performs:

1: update $\widehat{x}_j^i(t') = x_j(t)$ for $t' \geq t$
2: compute and maintain promise set $X_j^i(t')$ for $t' \geq t$
3: compute own control $u_i^{\text{team}}(t') = u_i^*(\widehat{x}_{\mathcal{N}}^i(t))$ for $t' \geq t$
4: compute first time $t^* \geq t$ such that (14) is satisfied
5: schedule information request to neighbors in $t^* - t$ seconds

(Respond to information request)
At any time t a neighbor $j \in \mathcal{N}(i)$ requests information, agent i performs:

1: send current state information $x_i(t)$ to agent j
2: send new promise $\mathcal{U}_i^j$ to agent j

(Event-triggered information update)
At all times t, agent i performs:

1: **if** there exists $j \in \mathcal{N}(i)$ such that $x_i(t) \notin X_i^j(t)$ **then**
2: send current state information $x_i(t)$ to agent j
3: send new promise $\mathcal{U}_i^j$ to agent j
4: **end if**

It is worth mentioning that the self-triggered approach described in Section III-C is a particular case of the team-triggered approach, where the promises are simply the guaranteed (or reachable) sets described in (11). In this scenario promises can never be broken so the event-triggered information updates never occur. Therefore, the class of team-triggered strategies contains the class of self-triggered strategies.

The following statement provides the main convergence result regarding distributed controller implementations synthesized via the team-triggered approach.

Proposition IV.2 *For the network model and problem setup described in Section III-A, if each agent executes the team-triggered coordination policy described in Algorithm 1, the state of the entire network asymptotically converges to the desired set D for all Zeno-free executions.*

Note that the result is only formulated for non-Zeno executions for simplicity of presentation. As discussed, the absence of Zeno behavior can be guaranteed via dwell times for the self- and event-triggered updates. In the following, we provide a proof sketch of this result that highlights its key components. We refer the interested reader to our work [28] that provides a detailed technical discussion that also accounts for the presence of dwell times in the event- and self-triggered updates.

Our first observation is that the evolution of the Lyapunov function V along the trajectories of the team-triggered coordination policy is nonincreasing by design. This is a consequence of the analysis of Section IV-B, which holds under the assumption that promises are never broken, together with the event-triggered updates described in Section IV-C, which guarantee that the assumption holds. Then, the main challenge to establish the asymptotic convergence properties of the team-triggered coordination policy is in dealing with its discontinuous nature. More precisely, since the information possessed by any given agent are sets for each of its neighbors, promises shrink to singletons when updated information is received. To make this point precise, let us formally denote by

$$S_i = \mathbb{P}^{\text{cc}}(\mathcal{X}_1) \times \cdots \times \mathbb{P}^{\text{cc}}(\mathcal{X}_{i-1}) \times \mathcal{X}_i \times \mathbb{P}^{\text{cc}}(\mathcal{X}_{i+1}) \times \cdots \times \mathbb{P}^{\text{cc}}(\mathcal{X}_N),$$

the state space where the variables that agent $i \in \{1, \ldots, N\}$ possesses live. Note that this set allows us to capture the fact that each agent i has perfect information about itself. Although agents only have information about their neighbors, the above space considers agents having promise information about all other agents to facilitate the analysis. This is only done to allow for a simpler technical presentation, and does not impact the validity of the arguments. The space the space that the state of the entire network lives in is then $S = \prod_{i=1}^{N} S_i$. The information possessed by all agents of the network at some time t is collected in

$$\big(X^1(t), \ldots, X^N(t)\big) \in S,$$

where $X^i(t) = (X^i_1(t), \ldots, X^i_N(t)) \in S_i$.

The team-triggered coordination policy corresponds to a discontinuous map of the form $S \times \mathbb{Z}_{\geq 0} \to S \times \mathbb{Z}_{\geq 0}$. The discontinuity makes it difficult to use standard stability methods to analyze the convergence properties of the network. Our approach to this problem consists of defining a set-valued map $M : S \times \mathbb{Z}_{\geq 0} \rightrightarrows S \times \mathbb{Z}_{\geq 0}$ with good continuity properties so that the trajectories of

$$(Z(t_{\ell+1}), \ell + 1) \in M(Z(t_\ell), \ell).$$

contain the trajectories of the team-triggered coordination policy. Although this 'over-approximation procedure' enlarges the set of trajectories to consider in the analysis, the gained benefit is that of having a set-valued map with suitable continuity properties that is amenable to set-valued stability analysis.

We conclude this section by giving an idea of how to define the set-valued map M, which is essentially guided by the way information updates can happen under the team-triggered coordination policy. Given $(Z, \ell) \in S \times \mathbb{Z}_{\geq 0}$, we define the $(N + 1)$th component of all the elements in $M(Z, \ell)$ to be $\ell + 1$ (this corresponds to evolution in time). The ith component of the elements in $M(Z, \ell)$ is the set given by one of following possibilities: (i) the case when agent i does not receive any new information in this step, in which case the promise sets available to agent i are simply propagated forward with respect to the promises made to it; (ii) the case when agent i has received information from at least one neighbor j, in which case the promise set X_j^i becomes an exact point $\{x_j\}$ and all other promise sets are propagated forward as in (i). Lumping together in a set the two possibilities for each agent gives rise to a set-valued map with good continuity properties: formally, M is closed (a set-valued map $T : X \rightrightarrows Y$ is closed if $x_k \to x$, $y_k \to y$ and $y_k \in T(x_k)$ imply that $y \in T(x)$). The proof then builds on this construction, combined with the monotonic evolution of the Lyapunov function along the trajectories of the team-triggered coordination policy, and employs a set-valued version of LaSalle Invariance Principle, see [25], to establish the asymptotic convergence result stated in Proposition IV.2.

V. Illustrative scenario: optimal robot deployment

In this section we illustrate the above discussion on self- and team-triggered design of distributed coordination algorithms in a scenario involving the optimal deployment of a robotic

sensor network. We start by presenting the original problem formulation from [29], where the authors provide continuous-time and periodic distributed algorithms to solve the problem. The interested reader is referred to [25] for additional references and a general discussion on coverage problems.

Consider a group of N agents moving in a convex polygon $S \subset \mathbb{R}^2$ with positions $P = (p_1, \ldots, p_N)$. For simplicity, consider single-integrator dynamics

$$\dot{p}_i = u_i, \tag{16}$$

where $\|u_i\| \leq v_{\max}$ for all $i \in \{1, \ldots, N\}$ for some $v_{\max} > 0$. The network objective is to achieve optimal deployment as measured by the following locational optimization function $\mathcal{H}$. The agent performance at a point q of agent i degrades with $\|q - p_i\|^2$. Assume a density $\phi : S \to \mathbb{R}$ is available, with $\phi(q)$ reflecting the likelihood of an event happening at q. Consider then the minimization of the function

$$\mathcal{H}(P) = E_\phi\big[\min_{i \in \{1,\ldots,N\}} \|q - p_i\|^2\big]. \tag{17}$$

This formulation is applicable in scenarios where the agent closest to an event of interest is the one responsible for it. Examples include servicing tasks, spatial sampling of random fields, resource allocation, and event detection.

With regards to the problem statement in Section III-A, $\mathcal{H}$ plays the role of the function V and the set D corresponds to the set of critical points of $\mathcal{H}$. Let us next describe the distributed continuous-time controller u^*. In order to do so, we need to recall some notions from computational geometry. A partition of S is a collection of N polygons $\mathcal{W} = \{W_1, \ldots, W_N\}$ with disjoint interiors whose union is S. The Voronoi partition $\mathcal{V}(P) = \{V_1, \ldots, V_N\}$ of S generated by the points P is

$$V_i = \{q \in S \mid \|q - p_i\| \leq \|q - p_j\|\,,\ \forall j \neq i\}.$$

When the Voronoi regions V_i and V_j share an edge, p_i and p_j are (Voronoi) neighbors. We denote the neighbors of agent i by $\mathcal{N}(i)$. For $Q \subset S$, the *mass* and *center of mass* of Q with respect to ϕ are

$$M_Q = \int_S \phi(q)dq, \quad C_Q = \frac{1}{M_Q}\int_Q q\phi(q)dq.$$

A tuple $P = (p_1, \ldots, p_N)$ is a centroidal Voronoi configuration if $p_i = C_{V_i}$, for all $i \in \{1, \ldots, N\}$. The controller u^* then simply corresponds to the gradient descent of the function $\mathcal{H}$, whose ith component is given by

$$-\frac{\partial \mathcal{H}}{\partial p_i} = 2M_{V_i}(C_{V_i} - p_i).$$

Remarkably, this continuous-time coordination strategy admits a periodic implementation without the need of determining any stepsize in closed form. This design is based on the observation that the function $\mathcal{H}$ can be rewritten in terms of the Voronoi partition as

$$\mathcal{H}(P) = \sum_{i=1}^{N} \int_{V_i} \|q - p_i\|^2 \phi(q) dq,$$

which suggests the generalization of $\mathcal{H}$ given by

$$\mathcal{H}(P, \mathcal{W}) = \sum_{i=1}^{N} \int_{W_i} \|q - p_i\|^2 \phi(q) dq, \tag{18}$$

where $\mathcal{W}$ is a partition of S, and the ith agent is responsible of the "dominance region" W_i. The function $\mathcal{H}$ in (18) is then to be minimized with respect to the locations P and the dominance regions $\mathcal{W}$. The key observations now are that: (i) for a fixed network configuration, the optimal partition is the Voronoi partition, i.e.,

$$\mathcal{H}(P, \mathcal{V}(P)) \leq \mathcal{H}(P, \mathcal{W}),$$

for any partition $\mathcal{W}$ of S, and (ii) for a fixed partition, the optimal agent positions are the centroids, in fact,

$$\mathcal{H}(P', \mathcal{W}) \leq \mathcal{H}(P, \mathcal{W}),$$

for any P', P such that $\|p_i' - C_{W_i}\| \leq \|p_i - C_{W_i}\|$, $i \in \{1, \ldots, N\}$. These two facts together naturally lead to a periodic implementation of the coordination policy u^* where agents repeatedly and synchronously acquire position information from their neighbors, update their own Voronoi cell, and move towards their centroid. Both the continuous-time and periodic implementations are guaranteed to make the network state asymptotically converge to the set of centroidal Voronoi configurations.

A. *Self-triggered deployment algorithm*

Here we describe a self-triggered implementation of the distributed coordination algorithm for deployment described above. There are two key elements in our exposition. The first is the notion of abstraction employed by individual agents about the behavior of their neighbors. In this particular scenario, this gives rise to agents operating with spatial partitions under uncertain information, which leads us to the second important element: the design of a controller that operates on set-valued information rather than point-valued information. Our exposition here follows [18].

Abstraction via maximum velocity bound: The notion of abstraction employed is based on the maximum velocity bound for (16). In fact, an agent knows that any other neighbor cannot travel farther than $v_{\max}\Delta t$ from its current position in Δt seconds. Consequently, the reachable set of an agent $i \in \{1, \dots, N\}$ is

$$\mathcal{R}_i(s, p_i) = \overline{B}(p_i, sv_{\max}) \cap S.$$

If t_ℓ denotes the time at which agent i has just received position information $p_j(t_\ell)$ from a neighboring agent $j \in \mathcal{N}(i)$, then the guaranteed set that agent i maintains about agent j is

$$\mathbf{X}_j^i(t) = \overline{B}(p_j(t_\ell), (t - t_\ell)v_{\max}) \cap S,$$

which has the property that $p_j(t) \in \mathbf{X}_j^i(t)$ for $t \geq t_\ell$. Figure 1(a) shows an illustration of this concept. Agent i stores the data in $\mathcal{D}^i(t) = (\mathbf{X}_1^i(t), \dots, \mathbf{X}_N^i(t)) \subset S^N$ (if agent i does not have

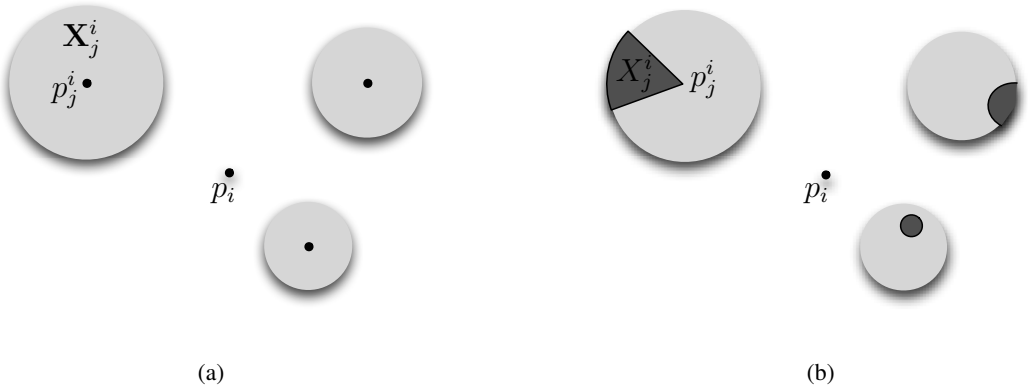


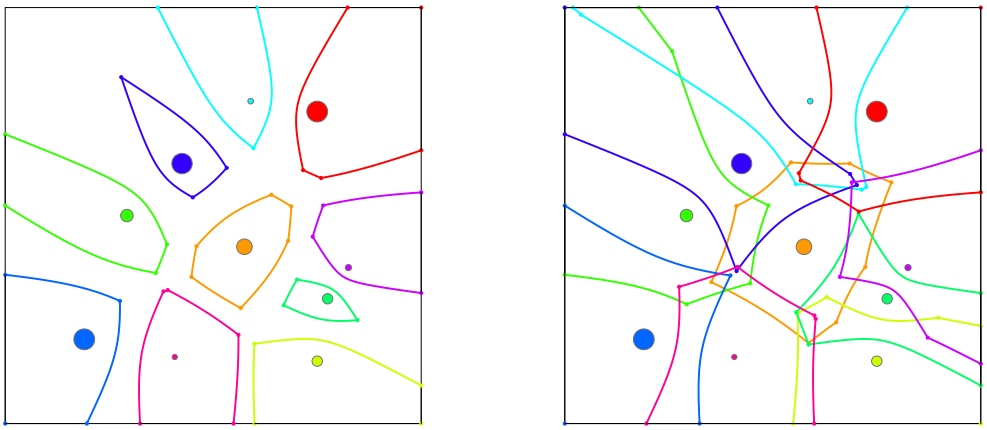
Fig. 1. Graphical representation of (a) guaranteed sets and (b) promise sets kept by agent i. In (b), dark regions correspond to the promise sets given the promises $\mathcal{U}_j^i$.

information about some agent j, we set $\mathbf{X}_j^i(t) = \emptyset$) and the entire memory of the network is $\mathcal{D} = (\mathcal{D}^1, \ldots, \mathcal{D}^N) \subset S^{N^2}$.

Design of controller operating on set-valued information: Agents cannot compute the Voronoi partition exactly with the data structure described above. However, they can still determine lower and upper bounds on their exact Voronoi cell, as we describe next. Given a set of regions $D_1, \ldots, D_N \subset S$, each containing a site $p_i \in D_i$, the guaranteed Voronoi diagram of S generated by $D = (D_1, \ldots, D_N)$ is the collection $\mathrm{g}\mathcal{V}(D_1, \ldots, D_N) = \{\mathrm{g}V_1, \ldots, \mathrm{g}V_N\}$,

$$\mathrm{g}V_i = \{q \in S \mid \max_{x \in D_i} \|q - x\| \leq \min_{y \in D_j} \|q - y\| \text{ for all } j \neq i\}.$$

The guaranteed Voronoi diagram is not a partition of S, see Figure 2(a). The important fact



(a) (b)

Fig. 2. Guaranteed (a) and dual guaranteed (b) Voronoi diagrams.

for our purposes is that, for any collection of points $p_i \in D_i$, $i \in \{1, \ldots, N\}$, the guaranteed Voronoi diagram is contained in the Voronoi partition, i.e., $\mathrm{g}V_i \subset V_i$, $i \in \{1, \ldots, N\}$. Each point in the boundary of $\mathrm{g}V_i$ belongs to the set

$$\Delta_{ij}^{\mathrm{g}} = \{q \in S \mid \max_{x \in D_i} \|q - x\| = \min_{y \in D_j} \|q - y\|\}, \tag{19}$$

for some $j \neq i$. Note that $\Delta_{ij}^{\mathrm{g}} \neq \Delta_{ji}^{\mathrm{g}}$. Agent p_j is a guaranteed Voronoi neighbor of p_i if $\Delta_{ij}^{\mathrm{g}} \cap \partial \mathrm{g}V_i$ is not empty nor a singleton. The set of guaranteed Voronoi neighbors of agent i is $\mathrm{g}\mathcal{N}_i(D)$. In our scenario, the 'uncertain' regions for each agent correspond to the guaranteed sets.

One can also introduce the concept of dual guaranteed Voronoi diagram of S generated by $D_1, \dots, D_N$, which is the collection of sets $\mathrm{dg}\mathcal{V}(D_1, \dots, D_N) = \{\mathrm{dg}V_1, \dots, \mathrm{dg}V_N\}$ defined by

$$\mathrm{dg}V_i = \{q \in S \mid \min_{x \in D_i} \|q - x\| \leq \max_{y \in D_j} \|q - y\| \text{ for all } j \neq i\}.$$

A similar discussion can be carried out for this notion, cf. [18]. The most relevant fact for our design is that for any collection of points $p_i \in D_i$, $i \in \{1, \dots, N\}$, the dual guaranteed Voronoi diagram contains the Voronoi partition, i.e., $V_i \subset \mathrm{dg}V_i$, $i \in \{1, \dots, N\}$.

The notions of guaranteed and dual guaranteed Voronoi cells allow us to synthesize a controller building on the observation that, if

$$\|p_i - C_{\mathrm{g}V_i}\| > \mathrm{bnd}_i \equiv \mathrm{bnd}(\mathrm{g}V_i, \mathrm{dg}V_i) = 2\,\mathrm{cr}(\mathrm{dg}V_i)\Big(1 - \frac{M_{\mathrm{g}V_i}}{M_{\mathrm{dg}V_i}}\Big), \tag{20}$$

then moving towards $C_{\mathrm{g}V_i}$ from p_i strictly decreases the distance $\|p_i - C_{V_i}\|$ (here, $\mathrm{cr}(Q)$ denotes the circumradius of Q). With this in place, an agent can actually move closer to the centroid of its Voronoi cell (which guarantees the decrease of $\mathcal{H}$), even when it does not know the exact locations of its neighbors. Informally, each agent moves as follows.

> *[Informal description]:* The agent moves towards the centroid of its guaranteed Voronoi cell until it is within distance bnd_i of it.

Note that, as time elapses without new information, the bound bnd_i grows until it becomes impossible to satisfy (20). This gives a natural triggering condition for when agent i needs updated information from its neighbors, which we describe informally as follows.

> *[Informal description]:* The agent decides that up-to-date location information is required if its computed bound in (20) is larger than a design parameter ε and the distance to the centroid of its guaranteed cell.

The role of ε is guaranteeing the lack of Zeno behavior: when an agent i gets close to $C_{\mathrm{g}V_i}$, this requires bnd_i to be small. The introduction of the design parameter is critical to ensure Zeno behavior does not occur. Note that each agent can compute the future evolution of bnd_i without updated information about their neighbors. This means that each agent can schedule exactly when the condition above will be satisfied, corresponding to a self-triggered implementation.

The overall algorithm is the result of combining the motion control law and the update decision policy with a procedure to acquire up-to-date information about other agents. We do not enter into

Algorithm 2: Self-triggered deployment algorithm

Agent $i \in \{1, \dots, N\}$ performs:

1: set $D = \mathcal{D}^i$
2: compute $L = \mathrm{g}V_i(D)$ and $U = \mathrm{dg}V_i(D)$
3: compute $q = C_L$ and $r = \mathrm{bnd}(L, U)$

(Update decision)

1: **if** $r \geq \max\{\|q - p_i\|, \varepsilon\}$ **then**
2: reset $\mathcal{D}^i$ by acquiring updated location information from Voronoi neighbors
3: set $D = \mathcal{D}^i$
4: set $L = \mathrm{g}V(D)$ and $U = \mathrm{dg}V(D)$
5: set $q = C_L$ and $r = \mathrm{bnd}(L, U)$
6: **end if**

(Motion control)

1: **if** $\|p_i - q\| > r$ **then**
2: set $u_i = v_{\max}\,\mathrm{unit}(q - p_i)$
3: **else**
4: set $u_i = 0$
5: **end if**

the details of the latter here and instead refer the reader to [18]. The self-triggered deployment algorithm is formally presented in Algorithm 2.

One can establish the same convergence guarantees for this algorithm as in the periodic or continuous case. In fact, for $\varepsilon \in [0, \mathrm{diam}(S))$, the agents' positions starting from any initial network configuration in S^n converges to the set of centroidal Voronoi configurations.

B. Team-triggered deployment algorithm

Here we describe a team-triggered implementation of the distributed coordination algorithm for optimal deployment. Given our previous discussion, the only new element that we need to specify with respect to the exposition of the self-triggered implementation in Section V-A is the notion of promises employed by the agents. For each $j \in \{1, \dots, N\}$, given the deployment controller $u_j^* : \prod_{i \in \mathcal{N}(j) \cup \{j\}} \mathbb{P}^{\mathrm{pcc}}(\mathcal{X}_i) \to \mathcal{U}_j$ and $\delta_j > 0$, the ball-radius control promise generated by agent j for agent i at time t is

$$\mathcal{U}_i^j = \overline{B}(u_j(X_{\mathcal{N}}^j(t)), \delta_j) \cap \mathcal{U}_j \qquad t' \geq t, \tag{21}$$

for some $\delta_j > 0$. This promise is a ball of radius δ_j in the control space $\mathcal{U}_j$ centered at the control signal used at time t. Here we consider constant δ_j for simplicity, but one can consider

time-varying functions instead, cf. [28]. Agent i stores the information provided by the promise sets in $\mathcal{D}^i(t) = (X_1^i(t), \ldots, X_N^i(t)) \subset S^N$. The event-triggered information updates, by which agents inform other agents if they decide to break their promises, take care of making sure that this information is always accurate, i.e., $p_i(t) \in X_i^j(t)$ for all $i \in \{1, \ldots, N\}$ and $j \in \mathcal{N}(i)$. The team-triggered deployment algorithm is formally presented in Algorithm 3 and has the same motion control law, update decision policy, and procedure to acquire up-to-date information about other agents as the self-triggered coordination one.

Algorithm 3: Team-triggered deployment algorithm

Agent $i \in \{1, \ldots, N\}$ performs:

set $D = \mathcal{D}^i$

compute $L = \mathrm{g}V_i(D)$ and $U = \mathrm{dg}V_i(D)$

compute $q = C_L$ and $r = \mathrm{bnd}(L, U)$

(Self-triggered update decision)

if $r \geq \max\{\|q - p_i\|, \varepsilon\}$ **then**

  reset $\mathcal{D}^i$ by acquiring updated location information from Voronoi neighbors

  set $D = \mathcal{D}^i$

  set $L = \mathrm{g}V(D)$ and $U = \mathrm{dg}V(D)$

  set $q = C_L$ and $r = \mathrm{bnd}(L, U)$

end if

(Event-triggered update decision)

if $p_i \notin X_i^j$ for any $j \in \mathcal{N}(i)$ **then**

  send updated position information p_i to agent j

  send new promise $\mathcal{U}_i^j$ to agent j

end if

(Motion control)

if $\|p_i - q\| > r$ **then**

  set $u_i = v_{\max}\,\mathrm{unit}(q - p_i)$

else

  set $u_i = 0$

end if

C. Simulations

Here we provide several simulations to illustrate the performance of the self-triggered and team-triggered deployment algorithms. We consider a network of $N = 8$ agents, moving in a 4m $\times$ 4m square, with a maximum velocity $v_{\max} = 1$m/s. The density ϕ is a sum of two Gaussian

functions

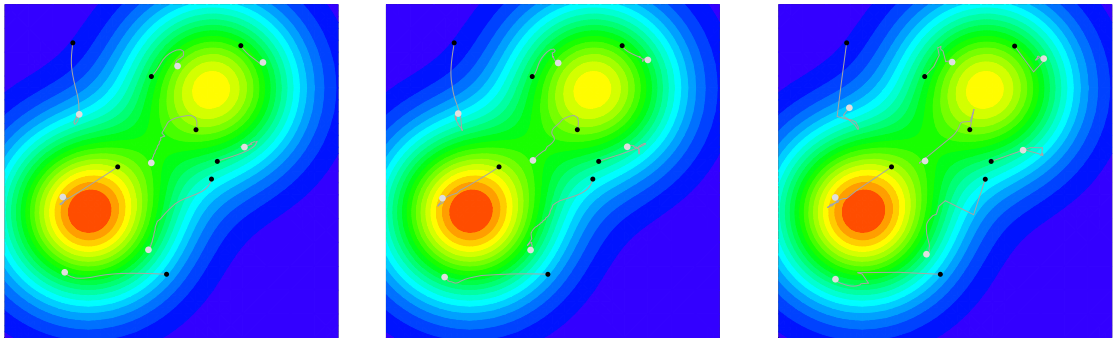
$$\phi(x) = 1.2e^{-\|x-q_1\|^2} + e^{-\|x-q_2\|^2},$$

with $q_1 = (1, 1.5)$ and $q_2 = (2.5, 3)$. We evaluate the total power consumed by communication among the agents during the execution of each algorithm. To do, we adopt the following model [30] for quantifying the total power $\mathcal{P}_i$ used by agent $i \in \{1, \ldots, 8\}$ to communicate, in $dBmW$ power units,

$$\mathcal{P}_i = 10 \log_{10} \Big[\sum_{j \in \{1,\ldots,N\}, i \neq j}^{n} \beta 10^{0.1 P_{i \to j} + \alpha \|p_i - p_j\|} \Big], \tag{22}$$

where $\alpha > 0$ and $\beta > 0$ depend on the characteristics of the wireless medium and $P_{i \to j}$ is the power received by j of the signal transmitted by i in units of $dBmW$. In our simulations, these values are set to 1.

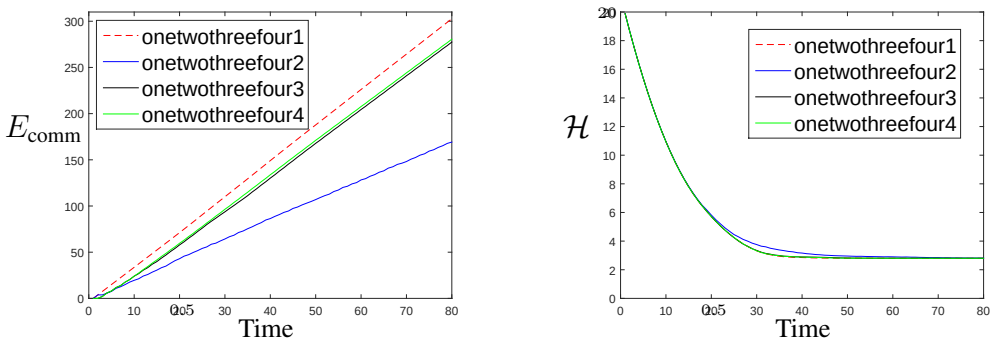
For the promises made among agents in the team-triggered strategy, we use $\delta_i = 2\lambda v_{\max}$ in (21), where $\lambda \in [0, 1]$ is a parameter that captures the 'tightness' of promises. Note that $\lambda = 0$ corresponds to exact promises, meaning trajectories can be computed exactly, and $\lambda = 1$ corresponds to no promises at all (i.e., recovers the self-triggered algorithm because promises become the entire allowable control set).



(a) Periodic (b) Self-triggered (c) Team-triggered

Fig. 3. Plot (a) is an execution of the communicate-at-all-times strategy [29], plot (b) is an execution of the self-triggered strategy and plot (c) is an execution of the team-triggered strategy with $\lambda = 0.5$. Black dots correspond to initial positions and light gray dots correspond to final positions.

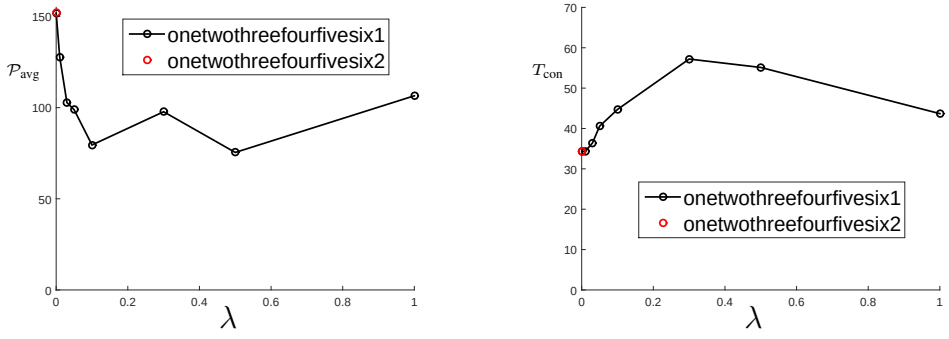
Figures 3 and 4 compare the execution of the periodic, self-triggered, and team triggered deployment strategies (the latter for two different tightnesses of promises, $\lambda = 0.25$ and $\lambda = 0.5$)



(a) Total energy expenditure (b) Objective function

Fig. 4. Plot (a) shows the total communication energy E_{comm} in Joules of executions of the communicate-at-all-times strategy, the self-triggered strategy, and the team-triggered strategy (for two different tightnesses of promises). Plot (b) shows the evolution of the objective function $\mathcal{H}$. All simulations have the same initial condition.

starting from the same initial condition. Figure 3 shows the algorithm trajectories while Figure 4(a) shows the total communication energy over time and Figure 4(b) shows the evolution of the objective function $\mathcal{H}$. These plots reveal the important role that the tightness of promises has on the algorithm execution. For instance, in Figure 4, one can see that, for $\lambda = 0.25$, the communication energy is reduced by half compared to the periodic strategy without compromising the network performance.



(a) (b)

Fig. 5. Average (a) communication power network consumption $\mathcal{P}_{\text{avg}}$ and (b) time to convergence T_{con} for the team-triggered deployment strategy with varying tightness of promise. Time to convergence is the time to reach 99% of the final convergence value of the objective function. For small λ, the amount of communication energy required is substantially reduced while the time to convergence only increases slightly.

In Figure 5 we explore what happens for various different tightnesses λ for the team-triggered strategy and compare them against the periodic and the self-triggered strategies. Figure 5(a)

shows the average power consumption in terms of communication energy by the entire network for varying levels of λ. Figure 5(b) shows the time it takes each execution to reach 99% of the final convergence value of the objective function $\mathcal{H}$. As we can see from these plots, for small $\lambda > 0$, we are able to drastically reduce the amount of required communication while only slightly increasing the time to convergence.

VI. Conclusions

This chapter has covered the design and analysis of triggered strategies for networked cyber-physical systems, with an emphasis on distributed interactions. At a fundamental level, our exposition aims to trade computation for less sensing, actuation, or communication effort by answering basic questions such as 'when to take state samples', 'when to update control signals', or 'when to transmit information'. We have focused on self-triggered approaches to controller design which build on the notion of abstraction of the system behavior to synthesize triggers that rely only on the current information available to the decision maker in scheduling future actions. The role of the notion of abstraction acquires special relevance in the context of networked systems, given the limited access to information about the network state possessed by individual agents. The often conservative nature of self-triggered implementations has motivated us to introduce the team-triggered approach, which allows us to combine the best components of event- and self-triggered control. The basic idea is to have agents cooperate with each other in providing better, more accurate abstractions to each other via promise sets. Such sets can then be used by individual agents to improve the overall network performance. The team-triggered approach opens up numerous venues for future research. Among them, we highlight the robustness under disturbances in the dynamics and sensor noise, more general models for individual agents, methods that tailor the generation of agent promises to the specific task at hand, methods for the systematic design of controllers that operate on set-valued information models, analytic guarantees on performance improvements with respect to self-triggered strategies (as observed in Section V-C), the impact of evolving topologies on the generation of agent promises, and the application to the synthesis of distributed strategies to other coordination problems.

Acknowledgments

This research was partially supported by NSF Award CNS-1329619.

REFERENCES

[1] M. Velasco, P. Marti, and J. M. Fuertes, "The self triggered task model for real-time control systems," in *Proceedings of the 24th IEEE Real-Time Systems Symposium*, pp. 67–70, 2003.

[2] R. Subramanian and F. Fekri, "Sleep scheduling and lifetime maximization in sensor networks," in *Symposium on Information Processing of Sensor Networks*, (New York, NY), pp. 218–225, 2006.

[3] W. P. M. H. Heemels, K. H. Johansson, and P. Tabuada, "An introduction to event-triggered and self-triggered control," in *IEEE Conf. on Decision and Control*, (Maui, HI), pp. 3270–3285, 2012.

[4] M. Mazo Jr. and P. Tabuada, "Decentralized event-triggered control over wireless sensor/actuator networks," *IEEE Transactions on Automatic Control*, vol. 56, no. 10, pp. 2456–2461, 2011.

[5] X. Wang and N. Hovakimyan, "L_1 adaptive control of event-triggered networked systems," in *American Control Conference*, (Baltimore, MD), pp. 2458–2463, 2010.

[6] M. C. F. Donkers and W. P. M. H. Heemels, "Output-based event-triggered control with guaranteed L_∞-gain and improved and decentralised event-triggering," *IEEE Transactions on Automatic Control*, vol. 57, no. 6, pp. 1362–1376, 2012.

[7] D. V. Dimarogonas, E. Frazzoli, and K. H. Johansson, "Distributed event-triggered control for multi-agent systems," *IEEE Transactions on Automatic Control*, vol. 57, no. 5, pp. 1291–1297, 2012.

[8] G. Shi and K. H. Johansson, "Multi-agent robust consensus-part II: application to event-triggered coordination," in *IEEE Conf. on Decision and Control*, (Orlando, FL), pp. 5738–5743, Dec. 2011.

[9] X. Meng and T. Chen, "Event based agreement protocols for multi-agent networks," *Automatica*, vol. 49, no. 7, pp. 2125–2132, 2013.

[10] Y. Fan, G. Feng, Y. Wang, and C. Song, "Distributed event-triggered control of multi-agent systems with combinational measurements," *Automatica*, vol. 49, no. 2, pp. 671–675, 2013.

[11] K. Kang, J. Yan, and R. R. Bitmead, "Cross-estimator design for coordinated systems: Constraints, covariance, and communications resource assignment," *Automatica*, vol. 44, no. 5, pp. 1394–1401, 2008.

[12] P. Wan and M. D. Lemmon, "Event-triggered distributed optimization in sensor networks," in *Symposium on Information Processing of Sensor Networks*, (San Francisco, CA), pp. 49–60, 2009.

[13] S. S. Kia, J. Cortés, and S. Martínez, "Distributed convex optimization via continuous-time coordination algorithms with discrete-time communication," *Automatica*, 2014. Submitted.

[14] D. Richert and J. Cortés, "Distributed linear programming with event-triggered communication," *SIAM Journal on Control and Optimization*, 2014. Submitted.

[15] A. Eqtami, D. V. Dimarogonas, and K. J. Kyriakopoulos, "Event-triggered strategies for decentralized model predictive controllers," in *IFAC World Congress*, (Milano, Italy), Aug. 2011.

[16] E. Garcia and P. J. Antsaklis, "Model-based event-triggered control for systems with quantization and time-varying network delays," *IEEE Transactions on Automatic Control*, vol. 58, no. 2, pp. 422–434, 2013.

[17] W. P. M. H. Heemels and M. C. F. Donkers, "Model-based periodic event-triggered control for linear systems," *Automatica*, vol. 49, no. 3, pp. 698–711, 2013.

[18] C. Nowzari and J. Cortés, "Self-triggered coordination of robotic networks for optimal deployment," *Automatica*, vol. 48, no. 6, pp. 1077–1087, 2012.

[19] X. Wang and M. D. Lemmon, "Event-triggered broadcasting across distributed networked control systems," in *American Control Conference*, (Seattle, WA), pp. 3139–3144, June 2008.

[20] M. Zhong and C. G. Cassandras, "Asynchronous distributed optimization with event-driven communication," *IEEE Transactions on Automatic Control*, vol. 55, no. 12, pp. 2735–2750, 2010.

[21] X. Wang and M. D. Lemmon, "Event-triggering in distributed networked control systems," *IEEE Transactions on Automatic Control*, vol. 56, no. 3, pp. 586–601, 2011.

[22] G. S. Seybotha, D. V. Dimarogonas, and K. H. Johansson, "Event-based broadcasting for multi-agent average consensus," *Automatica*, vol. 49, no. 1, pp. 245–252, 2013.

[23] M. Mazo Jr., A. Anta, and P. Tabuada, "On self-triggered control for linear systems: Guarantees and complexity," in *European Control Conference*, (Budapest, Hungary), Aug. 2009.

[24] C. Nowzari and J. Cortés, "Self-triggered optimal servicing in dynamic environments with acyclic structure," *IEEE Transactions on Automatic Control*, vol. 58, no. 5, pp. 1236–1249, 2013.

[25] F. Bullo, J. Cortés, and S. Martínez, *Distributed Control of Robotic Networks*. Applied Mathematics Series, Princeton University Press, 2009. Electronically available at http://coordinationbook.info.

[26] E. Garcia, Y. Cao, H. Yu, P. Antsaklis, and D. Casbeer, "Decentralised event-triggered cooperative control with limited communication," *International Journal of Control*, vol. 86, no. 9, pp. 1479–1488, 2013.

[27] C. Nowzari and J. Cortés, "Zeno-free, distributed event-triggered communication and control for multi-agent average consensus," in *American Control Conference*, (Portland, OR), pp. 2148–2153, 2014.

[28] C. Nowzari and J. Cortés, "Team-triggered coordination for real-time control of networked cyberphysical systems," *IEEE Transactions on Automatic Control*, 2013. Submitted.

[29] J. Cortés, S. Martínez, T. Karatas, and F. Bullo, "Coverage control for mobile sensing networks," *IEEE Transactions on Robotics and Automation*, vol. 20, no. 2, pp. 243–255, 2004.

[30] S. Firouzabadi, "Jointly optimal placement and power allocation in wireless networks," Master's thesis, University of Maryland at College Park, 2007.