Robotica (2018) volume 36, pp. 1077–1097.
doi:10.1017/S0263574718000231

Kernel Design and Distributed, Self-Triggered Control for Coordination of Autonomous Multi-Agent Configurations

Levi DeVries*, Aaron Sims and Michael D. M. Kutzer

Department of Weapons and Systems Engineering, United States Naval Academy, Annapolis, Maryland, 21402, USA. E-mails: amsims93@gmail.com, kutzer@usna.edu

(Accepted March 3, 2018. First published online: March 27, 2018)

SUMMARY

Autonomous multi-agent systems show promise in countless applications, but can be hindered in environments where inter-agent communication is limited. In such cases, this paper considers a scenario where agents communicate intermittently through a cloud server. We derive a graph transformation mapping the kernel of a graph's Laplacian to a desired configuration vector while retaining graph topology characteristics. The transformation facilitates derivation of a self-triggered controller driving agents to prescribed configurations while regulating instances of inter-agent communication. Experimental validation of the theoretical results shows the self-triggered approach drives agents to a desired configuration using fewer control updates than traditional periodic implementations.

KEYWORDS: Consensus-based control; Self-triggered control; Multi-agent control; Graph transformation; Asynchronous control.

1. Introduction

The expanding role of distributed multi-agent systems in military, civilian, and industrial applications has introduced interesting new questions in multi-agent control, coordination, and communication. Specifically, in bandwidth-limited or denied environments, there exists a need for control algorithms that drive agents to a desired system configuration while judiciously coordinating their control updates and time instances of agent interaction. These two objectives are typically at odds; increasing the time duration between control instances can slow convergence or cause system instability. This paper presents a novel approach to stabilizing multi-agent configurations by deriving a distributed controller that drives agents to a desired configuration and regulates agent control update instances to maintain stability of the desired configuration, an approach known as self-triggering.

We consider the problem of coordinating a multi-agent group to a desired system configuration assuming agents communicate over a specified network topology intermittently and indirectly through a server, which we refer to as a "cloud." The term "cloud" is proposed in this context in refs. [1] and [2] toward driving agents to consensus, where all agents converge to a common value. The concept of the cloud in this work shares similarity with that of a blackboard control architecture, where a "blackboard" component receives information and can store and disseminate information to agents, or sometimes referred to as nodes, upon request.[3] When an agent connects to the cloud it receives information about its neighbors from the last time they connected, implying that each agent must rely on old information to make its control decision. Each distributed agent then computes its control, uploads its current state information, disconnects, and executes the control for a chosen time interval.

This problem may arise in applications involving cyber-physical systems,[4] group dynamics,[5] coordination of unmanned vehicles,[2,6] or load balancing in distributed computing systems.[7] For example, communication is limited and costly in the undersea domain since signals can attenuate

* Corresponding author. E-mail: devries@usna.edu

at relatively short spatial scales. The authors in refs. [2] and [1] posed an example application coordinating surfacing times of underwater submarines when each vehicle can only receive and transmit information while surfaced. The goal of the controller is to coordinate surfacing times of agents and prescribe the desired control (for example, velocity) during the submerged interval. Toward addressing this problem, this paper considers the coordination of single-integrator kinematic agents to a desired configuration while simultaneously coordinating the instances in which they communicate with the cloud.

An extensive body of literature exists addressing the coordination of distributed multi-agent systems. A summative review of recent work is presented in ref. [8]. Generally speaking, research thrusts may be categorized into consensus algorithms, where all agents reach a common state,[9,10] distributed formation algorithms, where the group of agents (typically mobile vehicles) forms a predefined geometric configuration,[11–13] and triggering algorithms that focus on determining the times in which agents update their control.[14] This work derives a distributed formation controller that simultaneously coordinates agent's control update instances.

To drive agents to a desired configuration, the approach derived in this paper shares similarities with previous work in consensus-based formation control literature (e.g., see refs. [15] and [11], or [8] for an overview), where agents reach a desired configuration of states rather than a common state (called consensus). Like these works, we assume single-integrator agent kinematics and weight edges of the underlying agent communication graph Laplacian in the control to stabilize a desired agent configuration. In ref. [11], the author relates multi-agent kinematics to the advection equation and shows that carefully weighting communication in directed graph topologies results in stable, multi-agent formations. Similarly, the authors of ref. [12] derive edge weights to enforce a desired distance between communicating agents while simultaneously incorporating obstacle avoidance. In both approaches, however, the choice of weights affects the spectral properties of the graph and a systematic method of weight selection and allowable communication topologies for a desired formation is not specified. This paper extends the results of refs. [11] and [12] in that the graph transformation of this paper retains the spectral properties and communication topology of the inter-agent network and systematically generates the appropriate communication weights to achieve the desired configuration. In other related consensus-based formation work, the authors in refs. [16] and [15] incorporate a bias vector corresponding to the desired agent distance from the centroid of agent states. Achieving consensus relative to the centroid ensures convergence to the desired formation. This approach achieves convergence properties dictated by the spectral properties of the communication graph Laplacian, but requires each agent to know its required position relative to the centroid to achieve the desired formation. This paper achieves the same convergence properties but uses weights shared amongst neighboring agents to specify the formation rather than each agent requiring a known individual bias. Beyond consensus-based approaches to formation control, an extensive body of literature addresses formation control specific to non-holonomic mobile vehicles.[17–19]

A substantial amount of research has focused on coordinating the communication and control update instances of distributed systems. Generally speaking, the approaches fall into two categories. In event-triggered control, actuation and communication are triggered when a measured function, typically a measure of error, approaches some destabilizing threshold value.[20] This requires each agent to continuously monitor the error function to trigger control updates; further details regarding this approach and its applications can be found in refs. [14] and [21]. In self-triggered control, an agent forecasts its next control update time during its current one and maintains constant control between updates.[7,20,22,23] This paper uses the self-triggered approach. An agent connects to the cloud to download information about its neighbors. It then uploads its information to the cloud including the next time it will connect. Compared to traditional time-periodic implementations, event- and self-triggered control approaches can reduce the cumulative number of triggering instances required while remaining stable, with durations between triggers longer than allowed by time-periodic approaches.[2]

The self-triggered controller derived in this paper is motivated by refs. [1] and [2], but differs in that the results extend the self-triggering control to distributed configurations rather than consensus, where all agents reach a common state. Furthermore, it allows the cloud to reconfigure the formation of agents by "pushing" a new configuration to the group via inter-agent communication weightings. This work also shares similarity with ref. [24] where an event-triggered solution is proposed rather than a self-triggered one and with ref. [25] where the authors focus on achieving consensus subject to noise in the shared information between agents.

https://doi.org/10.1017/S0263574718000231 Published online by Cambridge University Press

The contributions of this work include (1) a theoretically justified kernel design technique that employs a similarity transformation to map the kernel vector of an N-dimensional graph Laplacian to a desired configuration vector with positive elements. The transformation presented in this work conserves spectral properties of the communication graph and allows the extensive body of research toward consensus to be extended to distributed formation problems. (2) We derive a decentralized continuous-time autonomous control algorithm steering agents to a desired time-invariant state configuration vector with positive elements using network communication defined by a prescribed graph, and (3) extend the continuous-time control to asynchronous, self-triggered control. Finally, (4) we provide experimental results using a network of direct current (DC) motor modules to validate the distributed, self-triggered consensus-based formation controller. The DC motors serve as placeholders for application specific "agents" such as mobile vehicles. Building on the experimental results, we simulate a group of autonomous quadrotors and use the multi-agent, self-triggered control algorithm to drive agents to a desired formation.

The remainder of the paper is organized as follows. Section 2 provides a review of relevant background information, including multi-agent kinematics, tools from graph theory, and properties of matrices used throughout the paper. Section 3 presents the primary theoretical results including a graph transformation algorithm and multi-agent control derivations of both conventional (i.e., continuous time) and self-triggered approaches. Section 4 provides an overview of experimental implementation and validation of the theoretical results from Section 3 using a network of DC motor modules communicating with the cloud over a serial interface. We follow the experimental results with a simulated application using the self-triggered controller to drive a group of quadrotor aircraft to a desired formation. Section 5 provides a summary of results and closing remarks.

2. Background

This section provides background information relevant to the theoretical results in Section 3. Section 2.1 discusses the kinematic model of agent motion, and Section 2.2 discusses the mathematical formulation of the agent communication network.

2.1. Kinematic motion model

Consistent with recent works in consensus-based multi-agent coordination and self-triggered control literature,[1,12,24] consider N agents with single-integrator kinematics. The position of the kth agent is $x_k \in \mathbb{R} \ \forall \ k = 1, \ldots, N$. The position of all agents is defined by the vector[(1)] $\mathbf{x} \in \mathbb{R}^N$ given by $\mathbf{x} = [x_1, \ldots, x_N]^{\mathrm{T}}$. The time derivative of each agent's position is controlled, giving the kinematic equation of motion for the kth agent

$$\dot{x}_k = u_k, \tag{1}$$

where u_k is a control variable. This paper derives state-feedback control algorithms $u_k = u_k(\mathbf{x})$, utilizing $\mathbf{x}$ or a subset of states within $\mathbf{x}$ to drive agents to a desired configuration denoted by $\mathbf{c} = [c_1, c_2, \ldots, c_N]^{\mathrm{T}}$. We assume limited communication between agents, implying that the kth agent's control may only depend on a subset of the state vector $\mathbf{x}$ as dictated by the underlying agent communication network.

2.2. Graph representation of agent communication

Inter-agent communication in multi-agent systems is often described using tools from graph theory.[26] Consider a graph, $\mathcal{G}$, composed of N vertices each representing an agent. Edges of the graph correspond to inter-agent communication channels; an edge from vertex n_i to vertex n_j indicates directed communication from agent i to agent j. The set of all neighbors of agent i is denoted $\mathcal{N}_i$. An undirected graph assumes bidirectional communication between agents whose vertices share an edge, whereas a directed graph, also called a digraph, specifies unidirectional information transfer between vertices sharing directed edges. Edges may be weighted to indicate relative weights of connection between agents. Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be an undirected, unweighted graph, where $\mathcal{V} = \{n_1, \ldots, n_N\}$ is the

[(1)]Bold fonts represent a column vector, e.g., $\mathbf{x} = [x_1, \ldots, x_N]^{\mathrm{T}}$ and capital letters represent an $M \times N$ matrix, e.g., $A \in \mathbb{R}^{M \times N}$.

https://doi.org/10.1017/S0263574718000231 Published online by Cambridge University Press

set of all vertices, and $\mathcal{E}$ is the set of all edges.[26] Assume any two vertices can be connected via a path along edges contained in $\mathcal{E}$. This implies that $\mathcal{G}$ is strongly connected.[26]

The matrices associated with the graph $\mathcal{G}$ are defined as follows. The unweighted adjacency matrix $A \in \mathbb{R}^{N\times N}$ of $\mathcal{G}$ represents $\mathcal{E}$ in matrix form. A is defined element-wise where $A_{i,i} = 0$ (ignoring an agent's ability to communicate with itself), and $A_{i,j} = 1$ if $(n_i, n_j) \in \mathcal{E}$ (assuming a single bidirectional communication channel between the ith and jth agents) and zero otherwise (see ref. [26] for further details). In a weighted graph $A_{i,j} = w_{i,j}$, where $w_{i,j}$ corresponds to a weight associated with the communication channel. For undirected graphs, A is symmetric. The degree matrix $D \in \mathbb{R}^{N\times N}$ is a diagonal matrix whose ith diagonal entry corresponds to the number of edges associated with the ith vertex.[26] From the adjacency and degree matrices, the graph Laplacian $L \in \mathbb{R}^{N\times N}$ is

$$L = D - A. \tag{2}$$

In an undirected graph, row and column sums of the Laplacian are zero implying that, for a strongly connected graph, the Laplacian has one zero eigenvalue whose corresponding eigenvector is the vector of ones $\mathbf{1} = [1, \ldots 1]^{\mathrm{T}} \in \mathbb{R}^N$. By definition, the kernel of L contains scalar multiples of $\mathbf{1}$, and the trivial zero vector.[26] Given the symmetry of the Laplacian associated with an undirected graph, the remaining eigenvalues are real and strictly greater than zero, with associated eigenvectors spanning the $N - 1$ dimensional complementary subspace of the vector $\mathbf{1}$.[26]

The graph transformation results of this paper assume an underlying graph that is undirected and strongly connected; we also utilize time-invariant communication topologies. However, the results are naturally extensible to time-varying communication topologies. The results of ref. [27] address stability and convergence with limited and time-varying communication topologies; these topics are beyond the scope of this work.

3. Theoretical Results

This section presents the theoretical contributions of the paper. Section 3.1 derives a kernel design technique that transforms the kernel of the graph Laplacian (2) while retaining its eigenvalue and agent communication characteristics. We show that the transformation maps an undirected graph to a weighted, directed graph whose weights are described by elements of the desired configuration. Section 3.2 utilizes the Laplacian transformation to derive a continuous-time control algorithm steering the multi-agent system (1) to a desired configuration. Section 3.3 extends the results of Section 3.2 by relaxing the continuous-time assumption to derive a distributed, self-triggered controller where agents periodically connect to a cloud server for updates regarding their neighbors' states.

3.1. Graph transformation

Tools from linear algebra enable derivation of a mapping that transforms the kernel of the graph Laplacian L to a desired configuration vector. The resulting transformed Laplacian matrix P has a specified kernel vector $\mathbf{c}$, such that $P\mathbf{c} = \mathbf{0}$.

Consider a similarity transform that produces similar Laplacian matrices[26] L and P. The similarity relationship is [28]

$$P = RLR^{-1}, \tag{3}$$

where $R \in \mathbb{R}^{N\times N}$ is an invertible matrix. By definition, similar matrices have equal eigenvalues. This implies that, for the Laplacian L of undirected graph $\mathcal{G}$, $\lambda_k(P) = \lambda_k(L) \geq 0$, where $\lambda_k(P)$ represents the kth eigenvalue of P.[26,28] Though the eigenvalues of P are equal to those of L, the eigenvectors of P are determined by the transformation matrix R. The following lemma shows that choosing

$$R = \mathrm{diag}\left(\frac{\mathbf{c}}{||\mathbf{c}||}\right), \tag{4}$$

where $c_k > 0, \forall k = 1, \ldots, N$, transforms the undirected graph Laplacian L into the Laplacian P of a weighted digraph with equivalent edges whose weights are determined by the desired configuration $\mathbf{c}$.[29] This paper considers desired configurations with strictly positive elements, though simulation results indicate the theoretical results remain valid for any configuration with non-zero elements.

https://doi.org/10.1017/S0263574718000231 Published online by Cambridge University Press

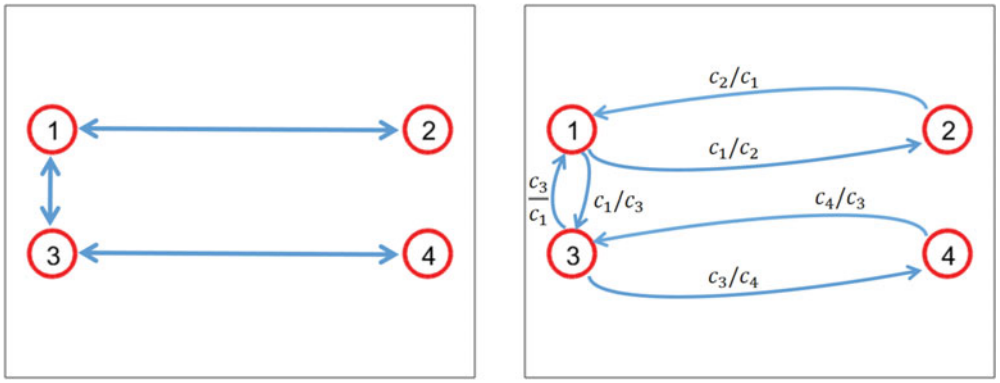

Fig. 1. (Left) Undirected graph $\mathcal{G}$ and (right) weighted digraph $\mathcal{G}^*$ produced by transformation (5).

Lemma 1. *The similarity transform (3) with transformation matrix (4) maps the Laplacian of an unweighted, undirected graph $\mathcal{G}$ to the Laplacian of a weighted, directed graph $\mathcal{G}^*$ whose edge from vertex n_i to n_j has weight c_i/c_j.*

Proof. Without loss of generality, assume $||\mathbf{c}|| = 1$. Since R and D are diagonal matrices, using (4) in (3) with (2) gives

$$P = RDR^{-1} - RAR^{-1} = D - RAR^{-1}. \tag{5}$$

Note that given the prescribed constraints on R, the degree matrix is unchanged. Define the transformed adjacency matrix $A^* \triangleq RAR^{-1}$. With R diagonal, the ith row of A is multiplied by c_i and the jth column of A is multiplied by $1/c_j$; zero elements remain unchanged. Therefore, $A^*_{i,j} = c_i/c_j$ corresponding to the edge between agents n_i and n_j has weight c_i/c_j, whereas $A^*_{j,i} = c_j/c_i$ has weight c_j/c_i. This implies edges i, j and j, i from A are retained in A^* but have reciprocal weights. By definition, P and A^* are, respectively, the Laplacian and adjacency matrices of the weighed digraph $\mathcal{G}^*$ with the same edges as $\mathcal{G}$ but directed edge i, j in $\mathcal{G}^*$ is weighted by c_i/c_j. □

Figure 1 illustrates the graph similarity transformation applied to a system of four agents. Figure 1(left) shows the undirected, unweighted graph $\mathcal{G}$, whereas Fig. 1(right) shows the similarity transformed graph $\mathcal{G}^*$ transformed by the desired configuration vector $\mathbf{c} = [c_1, c_2, c_3, c_4]^{\mathrm{T}}$. Note the inter-agent communication edges are equal; however, $\mathcal{G}^*$ weights directional communication between agents.

By definition, the similarity relation between Laplacians P and L preserves eigenvalues between the two matrices while changing the basis.[28] This change of basis allows the kernel of L to be mapped to a desired kernel of P through the relation $\mathbf{c} = \alpha R\mathbf{v}$ where equality holds since $\mathbf{v} = \mathbf{1}$. The non-zero term α is added for convenience to enable scaling in magnitude along the directions defined by $\mathbf{v}$ and $\mathbf{c}$. Invoking properties of the similarity transform gives the following.[29]

Theorem 1. *The transformed Laplacian (3), where L is the Laplacian matrix of a strongly connected, undirected graph $\mathcal{G}$ and R, given by (4) is an N-dimensional transformation matrix relating the kernel of L (defined as $\mathbf{v} = \mathbf{1}$) to the kernel of matrix P (defined as $\mathbf{c} \neq \mathbf{0}$), satisfies $P\mathbf{c} = \mathbf{0}$.*

Proof. Define $\mathbf{v} \in \ker(\mathrm{L})$, $||\mathbf{v}|| \neq 0$ and $\mathbf{c}$ such that $c_k > 0$, $\forall k = 1, \ldots, N$. Let R be defined by (4) such that it satisfies $\mathbf{c} = \alpha R\mathbf{v}$ for $\alpha = ||\mathbf{c}||/||\mathbf{v}||$. Multiplying $\mathbf{c}$ on the left by P and using (3) gives

$$P\mathbf{c} = RLR^{-1}\mathbf{c} = \alpha RLR^{-1}R\mathbf{v} = \alpha RL\mathbf{v} = \mathbf{0}, \tag{6}$$

since $\mathbf{v} \in \ker(L)$, thereby completing the proof. □

3.2. Continuous-time multi-agent configuration control

This section leverages the results of Section 3.1, using the transformed graph Laplacian P to derive a continuous time control algorithm steering agents to the desired configuration $\boldsymbol{c}$ while using inter-agent communication specified by the graph $\mathcal{G}$.

https://doi.org/10.1017/S0263574718000231 Published online by Cambridge University Press

Consider the potential function

$$V_a = \tfrac{1}{2}(R^{-1}\boldsymbol{x})^{\mathrm{T}} L (R^{-1}\boldsymbol{x}) = \tfrac{1}{2}\boldsymbol{x}^{\mathrm{T}} R^{-1} L R^{-1}\boldsymbol{x} \geq 0, \tag{7}$$

where R is diagonal and given by (4). Since R is diagonal with positive diagonal elements and L is positive semi-definite for an undirected, connected graph, the matrix $R^{-1}LR^{-1}$ is symmetric and positive semi-definite by construction. The time derivative of (7) with agent kinematics (1) is

$$\dot{V}_a = \tfrac{1}{2}\left(\boldsymbol{u}^{\mathrm{T}} R^{-1} L R^{-1}\boldsymbol{x} + \boldsymbol{x}^{\mathrm{T}} R^{-1} L R^{-1}\boldsymbol{u}\right). \tag{8}$$

Let the ith agent's control be

$$u_i = -K \sum_{j \in \mathcal{N}_i} \left(x_i - \frac{c_i}{c_j} x_j \right), \tag{9}$$

where $K > 0$ and $\mathcal{N}_i$ is the set of neighbors of agent i, such that the vector of agent control inputs is

$$\boldsymbol{u} = -KP\boldsymbol{x}, \tag{10}$$

where P is the transformed Laplacian defined by (5). The control (9) uses c_i, leading one to believe that *every* agent must be informed of its desired position and neighbor positions, which would seem to defeat the purpose of requiring agents to communicate. However, an agent need not know the *exact* desired position, only the ratio $w_{i,j} \triangleq c_i/c_j$ between it and its neighbors, which we henceforth refer to as the weight. Assuming agent i receives weights $w_{i,j}\ \forall j \in \mathcal{N}_i$ from its neighbors along with state information, an uninformed agent i need never exactly know c_i.

Using definition (5), Eq. (8) can be written as

$$\begin{aligned} \dot{V}_a &= -K/2\left(\boldsymbol{x}^{\mathrm{T}} P^{\mathrm{T}} R^{-1} L R^{-1}\boldsymbol{x} + \boldsymbol{x}^{\mathrm{T}} R^{-1} L R^{-1} P\boldsymbol{x}\right) \\ &= -K/2\boldsymbol{x}^{\mathrm{T}}(R^{-1}LRR^{-1}LR^{-1} + R^{-1}LR^{-1}RLR^{-1})\boldsymbol{x} \\ &= -K\boldsymbol{x}^{\mathrm{T}} R^{-1} L^2 R^{-1}\boldsymbol{x} \leq 0, \end{aligned} \tag{11}$$

since the quantity $R^{-1}L^2R^{-1}$ is symmetric and positive semi-definite as shown in the appendix. This implies the following.

Theorem 2. *Given a time-invariant configuration vector* $\mathbf{c} = [c_1, \ldots, c_N] \in \mathbb{R}^N$ *where* $c_i > 0$, $\forall i = 1, \ldots, N$ *and* $K > 0$, *all solutions of the multi-agent kinematics (1) with control (10) converge to the configuration* $\mathbf{x} = \alpha\mathbf{c}$ *for some constant* $\alpha \in \mathbb{R}$.

Proof. The potential function (7) is non-negative and proper in the reduced space of agent configurations relative to $\mathbf{c}$.[30] Since (7) is decreasing under control (10), the Invariance Principle stipulates that solutions converge to the largest invariant set for which $\dot{V}_a = 0$.[31] From (11), $\dot{V}_a = 0$ when $LR^{-1}\boldsymbol{x} = \mathbf{0}$. Note that

$$R^{-1}\boldsymbol{x} = \left[\frac{x_1}{c_1}, \ldots, \frac{x_N}{c_N}\right]^{\mathrm{T}},$$

implying that $LR^{-1}\boldsymbol{x} = 0$ only when

$$\frac{x_1}{c_1} = \frac{x_2}{c_2} = \cdots = \frac{x_N}{c_N}.$$

Thus, $\dot{V}_a = 0$ only when $\boldsymbol{x} = \alpha\boldsymbol{c}$ for some constant $\alpha \in \mathbb{R}$. Therefore, solutions converge to the configuration $\boldsymbol{x} = \alpha\boldsymbol{c}$. □

The control (10) drives agents to a scaled multiple of the desired configuration. To control agents to the *exact* configuration we introduce the concept of informed agents.[18] Let $a_i = 1$ if the ith agent has exact knowledge of c_i and zero otherwise. Define the knowledge vector $\boldsymbol{a} = [a_1, \ldots, a_N]^{\mathrm{T}}$ and let $\Lambda = \mathrm{diag}(\boldsymbol{a})$ be a diagonal matrix with a_i on the ith diagonal.

https://doi.org/10.1017/S0263574718000231 Published online by Cambridge University Press

We now augment the potential function to include configuration error terms only for those agents with information about their desired location. Consider the potential function

$$V = \tfrac{1}{2}\boldsymbol{e}^{\mathrm{T}}W\boldsymbol{e} \geq 0, \tag{12}$$

where $\boldsymbol{e} = \boldsymbol{x} - \boldsymbol{c}$ is the configuration error and $W = R^{-1}(L + \Lambda)R^{-1}$ is a symmetric matrix by construction and is positive definite if at least one agent is informed; proof showing W is positive definite is found in the appendix. Substituting the definition of the configuration error $\boldsymbol{e}$ with the fact that $R^{-1}LR^{-1}\boldsymbol{c} = 0$, we have

$$V = \tfrac{1}{2}\boldsymbol{e}^{\mathrm{T}}W\boldsymbol{e} = \tfrac{1}{2}\boldsymbol{x}^{\mathrm{T}}(R^{-1}LR^{-1})\boldsymbol{x} + \tfrac{1}{2}\boldsymbol{e}^{\mathrm{T}}R^{-1}\Lambda R^{-1}\boldsymbol{e}, \tag{13}$$

illustrating that (12) is simply the sum of (7) and a term corresponding to the square of the configuration error only for informed agents. The disagreement function is non-negative with respect to $\boldsymbol{e}$ with minimum occurring at $\boldsymbol{e} = \mathbf{0}$, implying $\boldsymbol{x} = \boldsymbol{c}$.

Given the agent kinematics (1), the time-derivative of (12) can be written as

$$\dot{V} = \boldsymbol{e}^{\mathrm{T}}R^{-1}(L + \Lambda)R^{-1}\dot{\boldsymbol{x}}, \tag{14}$$

motivating use of the vector of agent control inputs

$$\boldsymbol{u} = -K(P + \Lambda)\boldsymbol{e} = -KR(L + \Lambda)R^{-1}\boldsymbol{e} = -KP\boldsymbol{x} - K\Lambda\boldsymbol{e}, \tag{15}$$

where $K > 0$. Equation (15) implies the ith agent applies the control

$$u_i = -Ka_i\,(x_i - c_i) - K\sum_{j\in\mathcal{N}_i}\left(x_i - w_{i,j}x_j\right), \tag{16}$$

where $w_{i,j} = c_i/c_j$ is agent i's weight term shared by agent j.

Substituting (15) into (14) and simplifying give

$$\dot{V} = -K\boldsymbol{e}^{\mathrm{T}}Q\boldsymbol{e} \leq 0, \tag{17}$$

where

$$Q = R^{-1}(L + \Lambda)^2R^{-1}. \tag{18}$$

The matrix Q is symmetric and positive definite if $a_i = 1$ for at least one agent and positive semi-definite otherwise (see Lemma 4 in the appendix). Therefore, $\dot{V} \leq 0$ and equals zero only when $\boldsymbol{e} = \mathbf{0}$, which implies $\boldsymbol{x} = \boldsymbol{c}$ and the following theorem.

Theorem 3. *Given a time-invariant configuration vector* $\mathbf{c} = [c_1, \ldots, c_N] \in \mathbb{R}^N$ *where* $c_i > 0$ $\forall i = 1, \ldots, N$, $K > 0$, *and at least one informed agent, all solutions of the multi-agent kinematics (1) with control (16) converge to the configuration* $\mathbf{x} = \mathbf{c}$.

Proof. The potential function (13) is non-negative and proper in the reduced space of agent configurations relative to $\mathbf{c}$.[30] Since (13) is decreasing under control (15), the Invariance Principle stipulates that solutions converge to the largest invariant set where[31] $\dot{V} = 0$. Given that Q is positive definite, this occurs only when $\boldsymbol{e} = 0$ implying $\boldsymbol{x} = \boldsymbol{c}$. Therefore, solutions converge to the configuration $\boldsymbol{c}$. □

Figure 2 illustrates simulation of $N = 5$ agents with kinematics (1), control (16), and random initial conditions. The desired positions are uniformly distributed from twenty to sixty. Figure 2(left) shows the communication topology. Agent three is informed of the exact configuration; all other agents are uninformed. Figure 2(middle) shows agent positions over time. Solid lines represent agent positions, whereas dashed lines represent the desired configuration. Note all agents approach the desired configuration. Figure 2(right) shows the potential function (12) versus time, plotted on a log scale. Note the potential function (12) decreases toward zero.

https://doi.org/10.1017/S0263574718000231 Published online by Cambridge University Press

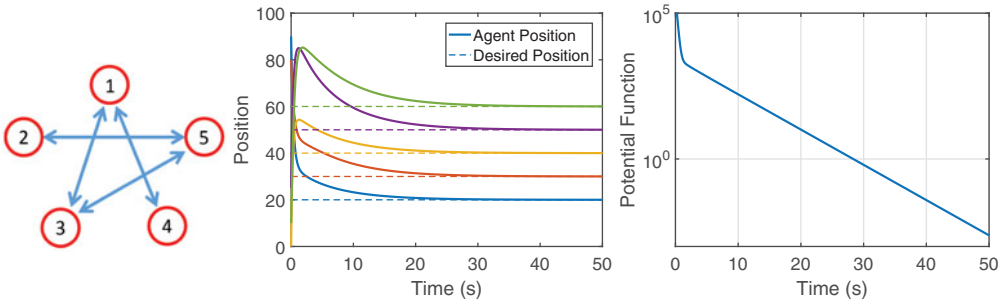

Fig. 2. $N = 5$ agents communicating according to the graph topology shown at (left) converging to a desired configuration (middle). (Right) The potential function (12) plotted vs. time.

Prior works[1,18] have proposed solving the distributed time-invariant formation problem with a variable transformation that defines the consensus variable $\boldsymbol{\xi} = \boldsymbol{x} - \boldsymbol{c}$. $\boldsymbol{\xi} \to \alpha \mathbf{1}$ is imposed using standard consensus control techniques. The resulting formation achieves desired relative spacing between agents with an arbitrary translational degree of freedom. The advantage of the graph transformation approach proposed in this paper is that it is equivalent to the consensus problem when $\boldsymbol{c} = \mathbf{1}$, it achieves the exact prescribed configuration in absolute coordinates, and it can simply extend consensus results to time-varying formation tracking. The authors' previous work[29] shows that the controller (16) can be adapted to track time-varying configurations $\boldsymbol{c}(t)$ with provably bounded configuration error. Additionally, this work extends the multitude of results addressing agent consensus problems to multi-agent formation problems by introducing the concept of a shared weight amongst agents. The following section shows a natural extension of self-triggered control to the multi-agent formation control problem.

3.3. Asynchronous self-triggered control

The results of the Section 3.2 assume that the agents update their control continuously. This requires all agents to communicate at every time step, which may increase bandwidth capacity requirements of the network. This section derives a distributed self-triggered control algorithm where each agent periodically connects to a cloud server. When an agent connects to the cloud it receives time-stamped state information about its neighbors and, if informed, about its desired state. The agent then computes its control and the next time it will connect to the cloud, uploads its information, and disconnects. Upon disconnecting from the cloud it applies its computed control for the specified interval of time until it re-connects.

The self-triggered control derivation in this section is inspired by the work in refs. [2] and [1] who derived a self-triggered, asynchronous control algorithm driving agents to a consensus state. This paper extends these works to steer the multi-agent system to a desired configuration and additionally allows the cloud to "push" desired configuration updates if desired. For consistency with previous works, we use the notation formulated in ref. [2].

When agent i makes its lth connection to the cloud at time t_i^l, it must compute its control $u(t_i^l)$ to apply over the interval $t \in \left[t_i^l, t_i^{l+1}\right)$ such that its contribution to the potential function (12) is negative over the interval. In order to compute its control using (16) and choose t_i^{l+1}, we assume the ith agent downloads:

1. The state of its neighbors at the last time they connected to the cloud, i.e., $x_j(t_j^{last})\ \forall j \in \mathcal{N}_i$
2. The control of its neighbors at the last time they connected to the cloud, $u_j(t_j^{last})\ \forall j \in \mathcal{N}_i$
3. The weight associated with its neighbors' interaction, $w_{i,j}\ \forall j \in \mathcal{N}_i$
4. The next time its neighbors will connect to the cloud, $t_j^{next}\ \forall j \in \mathcal{N}_i$
5. The time at which its neighbors' control will expire $t_j^{expire}\ \forall j \in \mathcal{N}_i$, at or prior to its next connection to the cloud
6. The maximum control its neighbors may apply over its current interval, referred to as the control promise of its neighbors, $M_j\ \forall j \in \mathcal{N}_i$
7. The desired position c_i only if it is an informed agent.

https://doi.org/10.1017/S0263574718000231 Published online by Cambridge University Press

After downloading the information and computing its control and next connecting time, the agent then uploads its

1. State, $x_i(t_i^l)$
2. Control, $u_i(t_i^l)$
3. Time of last connection, t_i^l
4. Time of next connection, t_i^{l+1}
5. Time of control expiration, t_i^{expire}
6. Control promise, M_i

We assume that the time required to connect, compute the necessary control quantities, and disconnect is negligible compared to the duration of the interval. Therefore, the goal is to use (12) to calculate the control $u_i(t_i^l)$, next connecting time t_i^{l+1}, and any necessary control expiration times t_i^{expire} to achieve the desired formation. The necessity for downloading and uploading these quantities will be justified in the derivation that follows.

Consider the potential function (12). At time t, the time derivative of (12) can be written

$$\dot{V}(t) = \sum_{i=1}^{N} \dot{x}_i(t) \left[a_i c_i^{-2} (x_i(t) - c_i) + W_i \boldsymbol{x}(t) \right] \tag{19}$$

$$= \sum_{i=1}^{N} u_i(t) \left[a_i c_i^{-2} (x_i(t) - c_i) + \sum_{j \in \mathcal{N}_i} c_i^{-2} x_i(t) - (c_i c_j)^{-1} x_j(t) \right], \tag{20}$$

where W_i is the ith row of W. Note that the ith term in the summation over i represents the contribution to the potential function due to agent i. By defining

$$\dot{V}_i(t) \triangleq u_i(t) \left[a_i (x_i(t) - c_i) + \textstyle\sum_{j \in \mathcal{N}_i} x_i(t) - w_{i,j} x_j(t) \right], \tag{21}$$

(19) becomes

$$\dot{V} = \sum_{i=1}^{N} c_i^{-2} \dot{V}_i. \tag{22}$$

The potential can then be written as

$$V(\boldsymbol{x}(t)) = V(\boldsymbol{x}(0)) + \int_0^t \dot{V}(\boldsymbol{x}(\tau)) \mathrm{d}\tau = V(\boldsymbol{x}(0)) + \sum_{i=1}^{N} c_i^{-2} \int_0^t \dot{V}_i(\boldsymbol{x}(\tau)) \mathrm{d}\tau. \tag{23}$$

The goal of the self-triggering algorithm is for the ith agent to compute intervals $t \in \left[t_i^l, t_i^{l+1} \right)$, for discrete cloud communication instances $l = 1, 2, \ldots$ over which it can apply the constant control $u_i(t_i^l)$ such that $\dot{V}_i(t) \leq 0$.

When agent i downloads information from the cloud at time t_i^l, it is able to update its understanding of the state of a neighbor up to that neighbor's next cloud connection, t_j^{next}. Therefore, for $t \in [t_i^l, t_j^{next})$

$$x_j(t) = x_j(t_j^{last}) + u_j(t_j^{last}) \left(t - t_j^{last} \right). \tag{24}$$

Consistent with ref. [2], define $T_i = \min_{j \in \mathcal{N}_i} t_j^{next}$ as the first time a neighbor will re-connect to the cloud. Agent i can compute agent j's state exactly until T_i. In the interval $t_i^l \leq t \leq T_i$ the ith agent's contribution to the potential function derivative (19) is

$$\dot{V}_i = u(t_i^l) \left[a_i (x_i - c_i) + c_i^2 W_i \boldsymbol{x}(t_i^l) + (t - t_i^l) \left(a_i u(t_i^l) + c_i^2 W_i \boldsymbol{u}(t_i^l) \right) \right]. \tag{25}$$

https://doi.org/10.1017/S0263574718000231 Published online by Cambridge University Press

Let t^* be the smallest time for which (25) is less than zero. If $t^* < T_i$, then agent i sets its ideal next time to connect to the cloud $t_i^{ideal} = t^*, t_i^{expire} = t_i^{ideal}$, uploads its information to the cloud, and disconnects. Note, under certain circumstances $t_i^{l+1} = t_i^{ideal}$; however, cases exist in which $t_i^{l+1} \neq t_i^{ideal}$, which are discussed in Section 3.4.

Note that the term

$$c_i^2 W_i \boldsymbol{x}(t_i^l) = c_i^2 \sum_{j \in \mathcal{N}_i} c_i^{-2} x_i(t_i^l) - (c_i c_j)^{-1} x_j(t_i^l) = \sum_{j \in \mathcal{N}_i} x_i(t_i^l) - w_{i,j} x_j(t_i^l)$$

in (25) is used as shorthand and does not require the ith agent to have knowledge of the full state vector $\boldsymbol{x}(t_i^l)$ or c_i. Uninformed agents calculate t^* using

$$u(t_i^l)\left[\left(\sum_{j \in \mathcal{N}_i} x_i(t_i^l) - w_{ij} x_j(t_i^l)\right) + (t - t_i^l)\left(\sum_{j \in \mathcal{N}_i} u_i(t_i^l) - w_{ij} u_j(t_j^{last})\right)\right] \leq 0. \tag{26}$$

If $t^* > T_i$, agent i can no longer precisely predict its contribution to (19). Agent i's projected state is

$$x_i(t) = x_i(T_i) + u_i(t_i^l)\,(t - T_i)\,. \tag{27}$$

Moreover, for $t > T_i$, agent i's understanding of its neighbor j's state is

$$x_j(t) = x_j(T_i) + \int_{T_i}^{t} \dot{x}_j(\tau)\mathrm{d}\tau. \tag{28}$$

Following refs. [2] and [4], we assume that the jth agent uploads a promise $M_j(t) \geq 0$ that its control will not exceed

$$\left|\dot{x}_j(t)\right| = |u_j(t)| \leq M_j(t).$$

By downloading the promised value from agent j, agent i can predict bounds on the state of agent j for $t > T_i$ given by

$$x_j(T_i) - (t - T_i)M_j(t) \leq x_j(t) \leq x_j(T_i) + (t - T_i)M_j(t). \tag{29}$$

The process of choosing a promise $M_j(t)$ is described in Section 3.4. Substituting (27) and (28) into (21) gives

$$\begin{aligned}\dot{V}_i(t) = u_i(t_i^l)\left[a_i(x_i(T_i) - c_i) + c_i^2 W_i \boldsymbol{x}(t_i^l) + a_i u_i(t_i^l)(t - t_i^l)\right] \\ + u_i(t_i^l)\left[\sum_{j \in \mathcal{N}_i} u_i(t_i^l) - w_{i,j}\int_{t_i^l}^{t} \dot{x}_j(\tau)\mathrm{d}\tau\right].\end{aligned} \tag{30}$$

Using (30) with the bounding relationship (29) reveals the interval of time upon which agent i can ensure $\dot{V}_i \leq 0$. If $u(t_i^l) < 0$, the sufficient condition for $\dot{V}_i \leq 0$ is

$$(t - T_i)\left[a_i u_i(t_i^l) + \sum_{j \in \mathcal{N}_i} u_i(t_i^l) - w_{ij} M_j(t)\right] \geq a_i\,(c_i - x_i(T_i)) + \sum_{j \in \mathcal{N}_i} x_i(T_i) - w_{ij} x_j(T_i). \tag{31}$$

Similarly, if $u(t_i^l) > 0$

$$(t - T_i)\left[a_i u_i(t_i^l) + \sum_{j \in \mathcal{N}_i} u_i(t_i^l) + w_{ij} M_j(t)\right] \leq a_i\,(c_i - x_i(T_i)) + \sum_{j \in \mathcal{N}_i} x_i(T_i) - w_{ij} x_j(T_i). \tag{32}$$

https://doi.org/10.1017/S0263574718000231 Published online by Cambridge University Press

Define T_i^* as the smallest time $T_i^* > T_i$ that condition (31) or (32) is no longer satisfied. By defining T_i^* in this fashion, agent i guarantees its contribution to the potential function $\dot{V}_i \leq 0$ for all times in the interval $t \in \left[t_i^l, T_i^*\right]$.

We allow small intervals where $\dot{V}_i$ can be positive so long as the overall contribution over the interval $[t_i^l, t_i^{l+1}]$ remains negative;[1,2] specifically,

$$\Delta V_i^l \triangleq \int_{t_i^l}^{t_i^{l+1}} \dot{V}_i(\tau)\mathrm{d}\tau < 0. \tag{33}$$

To compute the allowed time in which the contribution to $\dot{V}_i$ can be positive, partition the interval $[t_i^l, t_i^{l+1})$ into the precisely known and unknown, but bounded portions. Let B_i be the contribution to the potential function over the interval from t_i^l to T_i in which the ith agent can exactly predict its contribution to the overall potential[2]

$$B_i = \int_{t_i^l}^{T_i} \dot{V}_i(\tau)\mathrm{d}\tau, \tag{34}$$

and let C_i be the contribution when the ith agent is no longer certain of its contribution to the potential function

$$C_i = \int_{T_i}^{t} \dot{V}_i(\tau)\mathrm{d}\tau. \tag{35}$$

Using (28) and (30), one can show C_i is bounded such that

$$C_i \leq \bar{C}_i = \gamma(t - T_i) + \tfrac{\beta}{2}(t - T_i)^2, \tag{36}$$

where

$$\gamma = u(t_i^l)\left[a_i\left(x_i(T_i) - c_i\right) + \sum_{j\in\mathcal{N}_i} x_i(T_i) - w_{ij}x_j(T_i)\right], \tag{37}$$

and

$$\beta = \left[a_i u_i\left(t_i^l\right)^2 + \sum_{j\in\mathcal{N}_i} u_i\left(t_i^l\right)^2 + w_{ij}M_j(t)\left|u_i\left(t_i^l\right)\right|\right]. \tag{38}$$

We can then write[1]

$$\Delta V_i^l = B_i + C_i \leq B_i + \bar{C}_i(t).$$

This leads to defining T_i^{total} as the smallest time $T_i^{total} \geq T_i$ where

$$B_i + \bar{C}_i(T_i^{total}) < 0, \tag{39}$$

is no longer satisfied. The time T_i^{total} is the ith agent's prediction of the longest time it can remain disconnected from the cloud and still make a beneficial contribution to the potential function (12).

The ith agent can balance its choice of t_i^{l+1} between ensuring $\dot{V}_i$ remain negative over $[t_i^l, t_i^{l+1}]$ ($t_i^{l+1} = T_i^*$) or that its overall contribution to the potential function is negative over $[t_i^l, t_i^{l+1}]$ ($t_i^{l+1} < T_i^{total}$). Define $\sigma_i \in [0, 1]$ as a parameter weighting the balance between the two choices and let[1,2]

$$t_i^{ideal} = (1 - \sigma_i)\, T_i^* + \sigma_i T_i^{total}. \tag{40}$$

https://doi.org/10.1017/S0263574718000231 Published online by Cambridge University Press

Using $\sigma_i = 0$ ensures $\dot{V}_i < 0$ for the entire interval, which results in faster convergence to the desired configuration but more triggers over a given time span. Conversely, choosing $\sigma_i = 1$ causes (12) to decrease at the slowest possible rate but presumably requires fewer triggers over a given time span.

3.4. Selecting and implementing promises and avoiding Zeno behavior

Selection of the next triggering time t_i^{l+1} relies on knowledge of promises from neighboring agents, $M_j(t)$. This section stipulates when $t_i^{l+1} = t_i^{ideal}$ from the previous section or when a modification to t_i^{l+1} and $u_i(t_i^l)$ must be applied. Additionally, the derivation in Section 3.3 guarantees the potential function is strictly decreasing, but does not guarantee an agent will not need to reconnect an infinite number of times in a finite interval, known as Zeno behavior.[14] Fortunately, the method of choosing promises and avoiding Zeno behavior developed by Bowman et al.[2] and its precursor work by Nowzari and Pappas[1] for consensus is equivalently applicable to the distributed consensus-based formation control approach in this paper. An overview of the method derived in ref. [2] follows. For detailed discussion, the interested readers are referred to refs. [2], [1], and [4].

When it connects to the cloud an agent makes promise M_i about its control over the interval $t_i^{l+1} - t_i^l$. Equations (31), (32), and (39) depend on promises made between agents and dictate the duration of time an agent may be disconnected from the cloud. Larger promises produce shorter intervals and faster convergence, whereas smaller promises enable longer intervals but may slow the rate of convergence. To balance these effects, assume agent i chooses its promise based on its ideal control,[2]

$$M_i = |u_i^*(t_i^l)|, \tag{41}$$

where $u_i^*(t_i^l)$ is given by (16). During the interval an agent is disconnected it must abide by promises it has made to its neighbors. Therefore, if the promise calculated when an agent connects is greater than the promise it made over the previous interval, it must limit its control until all neighbors are aware of its latest promise.

Define

$$\tau_{ij}^l = t_j^{next}(t_i^l), \tag{42}$$

as the time that neighbor j is made aware of agent i's promise and

$$\xi_{ij} = \arg\max_{l:\tau_{ij}^l \le t} \tau_{ij}^l, \tag{43}$$

the index of the most recent promise made by agent i to its neighbor j. The most recent promise made by agent i to agent j is then $M_i(\tau_{i\xi_{ij}})$. Let the set of all promise indices made to all neighbors of i on the interval $[t_i^l, t_i^{l+1}]$ be defined[2]

$$\mathcal{P}_i^l = \{\xi_{ij} \mid j \in \mathcal{N}_i, t \in \left[t_i^l, t_i^{l+1}\right)\}. \tag{44}$$

Given the set of promises $\mathcal{P}_i^l$, agent i's control must be bounded by the minimum of all promises made to its neighbors,

$$u_i^{max}(t_i^l) = \min_{k \in \mathcal{P}_i^l} M(\tau_{i\xi_{ik}}). \tag{45}$$

The control that agent i must apply is then given in piecewise form as

$$u(t_i^l) = \begin{cases} u(t_i^l) & |u_i^*(t_i^l)| \le u_i^{max}(t_i^l) \\ u_i^{max}(t_i^l)\frac{u_i^*(t_i^l)}{|u_i^*(t_i^l|} & \text{otherwise.} \end{cases} \tag{46}$$

Thus, if all agents apply (46), each will abide by their promises, implying that agent states will converge to the desired configuration $\boldsymbol{c}$.

To avoid Zeno behavior, the authors in refs. [2] and [4] introduce a fixed dwell time $T_{dwell} > 0$ and ensure that agents all agents abide by $t_i^{l+1} \ge t_i^l + T_i^{dwell}$ for all i and l. Agents must remain

https://doi.org/10.1017/S0263574718000231 Published online by Cambridge University Press

disconnected from the cloud for at least T_i^{dwell}. This implies that if $t_i^* < T_i^{dwell}$, it may be impossible for an agent to remain disconnected from the cloud for a duration of T_i^{dwell} yet contribute negatively to the potential function.

To ensure an agent does not increase the potential function, note in Eq. (21) that if $u_i(t_i^l)$ is set to zero (the agent remains still), then $\dot{V}_i = 0$. If $t_i^{ideal} < T_i^{dwell}$, then one can partition the interval into the time until agent i can no longer contribute a decrease to the potential function, followed by a time that must pass before T_i^{dwell} has elapsed and the agent can trigger again. Define t_i^{expire} as the interval of time between when an agent triggers and when its contribution to the potential function is zero. Then, let the control be defined piecewise such that $u(t)$ is given by (46) for $t \in [t_i^l, t_i^{expire})$ and $u(t) = 0$ for $t \in [t_i^{expire}, t_i^l + T_i^{dwell})$. To summarize, if $t_i^{ideal} < T_i^{dwell}$, set $t_i^{l+1} = t_i^l + T_i^{dwell}$ and $t_i^{expire} = t_i^{ideal}$. Otherwise, $t_i^{l+1} = t_i^{ideal}$.

An overview of the self-triggered control algorithm is shown in Algorithm 1. Because the results of Section 3.1 directly relate the multi-agent formation problem to the multi-agent consensus problem, Algorithm 1 shares similarity with that proposed in ref. [2] with the exception of the quantities used to compute the triggering times and the additional configuration weights $w_{i,j} = c_i/c_j$ shared amongst agents. The following theorem proves the result of Sections 3.3 and 3.4.

Theorem 4. *Given a time-invariant configuration vector* $\mathbf{c} = [c_1, \ldots, c_N]$ *where* $c_i \neq 0 \ \forall i = 1, \ldots, N$, $K > 0$, *and at least one informed agent, all solutions of the multi-agent kinematics (1) with control and triggering times specified by Algorithm 1 converge to the configuration* $\mathbf{x} = \mathbf{c}$.

Proof. Proof of the theorem follows similarly to ref. [2] with potential function given by (12). For brevity, an outline of the proof is provided here. The potential function (12) is non-negative for all $x \in \mathbb{R}$. Algorithm 1 stipulates that the ith agent's contribution to (12) in the time interval $\left[t_i^l, t_i^{l+1}\right)$ is non-positive, implying that (12) is decreasing. Specifically, the selection of $t^{ideal} < T^{total}$ ensures $\Delta V_i < 0$. Furthermore, if $t^{ideal} < T^{dwell}$, the control expiration time choice t^{expire} ensures the ith agents contribution to the potential is zero for the remainder of the interval. Since the interval $\left[t_i^l, t_i^{l+1}\right)$ is greater than or equal to T^{dwell}, Zeno behavior cannot be achieved.

Since (12) is non-negative in the reduced space of agent configurations relative to $\mathbf{c}$ and decreasing under control stipulated by Algorithm 1, the Invariance Principle[31] stipulates that solutions converge to the largest invariant set where $\dot{V} = 0$. This occurs only when $\mathbf{x} = \mathbf{c}$. Therefore, solutions converge to the configuration $\mathbf{x} = \mathbf{c}$. □

Figure 3 illustrates simulation of $N = 5$ agents with the equations of motion (1) implementing the asynchronous self-triggered control algorithm detailed in Algorithm 1 for $t = 75$ s. The initial condition of each agent is equal to those in Fig. 2 for comparison purposes. Agents communicate via the cloud interface and have neighbors specified by the graph shown in Fig. 2(left). Each agent has gain $K = 0.5$ and tuning parameter $\sigma_i = 0.5$. Figure 3(top-left) shows agent positions over time. Figure 3(top-right) shows the control inputs. Figure 3(bottom-left) and (bottom-right) shows the potential function (12) and triggering instances over time, respectively. Note that agents converge to the desired configuration as indicated in Fig. 3(top-left) and (bottom-left). The cumulative number of triggering instances is illustrated by the solid black line in Fig. 3(bottom-right); individual agent triggering instances are also shown. Agent three is informed as indicated in the figure. Note that the informed agent has the most triggering instances in this case. However, over many trials, numerical simulations suggest the graph topology and initial conditions influence triggering totals more than informed status.

4. Experimental Results and Simulated Application

The distributed self-triggered control result of Section 3.3 may be beneficial to many applications requiring the coordination of positions, orientations, power production, or computational resource distribution of agents to a prescribed configuration. For example, Algorithm 1 can be used to orient an array of solar reflectors distributed over large spatial distances to concentrate light for power or heat generation, or for coordinating the positions of mobile vehicles incapable of direct communication. For unmanned underwater vehicles operating on large spatiotemporal scales where the complex vehicle dynamics can be approximated by a holonomic motion model, the self-triggering algorithm specifies agent surfacing times and ensuing velocities over the next submerged interval. More generally, the

https://doi.org/10.1017/S0263574718000231 Published online by Cambridge University Press

Algorithm 1 Self-triggered coordination to a specified configuration

At time t_i^l agent $i \in \{1, \ldots, N\}$ does the following:

if Agent i is informed **then**
  Download $t_j^{last}, t_j^{expire}, t_j^{next}, x_j(t_j^{last}), u_j(t_j^{last}), M_j, w_{ij}$ for all $j \in \mathcal{N}_i$ and a_i and c_i from the cloud
else
  Download $t_j^{last}, t_j^{expire}, t_j^{next}, x_j(t_j^{last}), u_j(t_j^{last}), M_j, w_{ij}$ for all $j \in \mathcal{N}_i$ from the cloud
end if
Update neighbor positions to current time using (24)
Compute ideal control $u_i^*(t_i^l) = -Ka_i(x_i(t_i^l) - c_i) - K\sum_{j\in\mathcal{N}_i} x_i(t_i^l) - w_{ij}x_j(t_i^l)$ ignoring the first term if uninformed
Compute $u_i^{max}(t_i^l)$ using (45) and (42)
Compute applied control $u_i(t_i^l)$ using (46)
Compute soonest connection time by all neighbors $T_i = \min_{j\in\mathcal{N}_i} t_j^{next}$
Compute t^* using (25) as first time when $\dot{V}_i \leq 0$ is not satisfied
if $t^* < T_i$ **then**
  Set $t_i^{ideal} = t^*$
else
  if $u(t_i^l) = 0$ **then**
    Set $t_i^{l+1} = T_i + T_{dwell}$
  else
    if $u(t_i^l) < 0$ **then**
      Compute T_i^* as the soonest time when (31) is no longer satisfied
    else
      Compute T_i^* as the soonest time when (32) is no longer satisfied
    end if
    Compute B_i using (34)
    Compute T_i^{total} as the first time when (39) is no longer satisfied
    Set t_i^{ideal} using (40)
  end if
end if
if $t_i^{ideal} < t_i^l + T_i^{dwell}$ **then**
  Set $t^{expire} = t_i^{ideal}$
  Set $t_i^{l+1} = t_i^l + T_i^{dwell}$
else
  Set $t_i^{expire} = t_i^{l+1} = t_i^{ideal}$
end if
Upload promise (41) to the cloud
Upload $t_i^{last} = t_i^l$, $t_i^{next} = t_i^{l+1}$, t_i^{expire}, $u(t_i^l)$, and $x_i(t_i^l)$ to cloud
Disconnect from the cloud, set $u_i(t) = u(t_i^l)$ for $t \in [t_i^l, t_i^{expire})$, $u_i(t_i^l) = 0$ for $t \in [t_i^{expire}, t_i^{l+1})$

self-triggered control output specifying a necessary velocity can be used as a reference for coordination of unmanned vehicles with traditionally non-linear dynamics.[12] As a proof of concept, the experimental results in this section use DC motor modules as generic "agents" whose angular velocity is specified by the distributed, self-triggered controller in Algorithm 1. Building on the experimental results, we follow with a simulation applying the self-triggered controller to drive a group of unmanned quadrotors to a desired formation.

4.1. Experimental results: DC motor control

To validate the proposed self-triggered control algorithm of Section 3.3, eight DC motor modules comprised of a 22-V DC Motor (GMX-6MP009A, Matsushita Electric) with an optical position encoder (1600 pulses per revolution after 8:1 gearing), an embedded microcontroller (mbed LPC1768, ARM mbed), and a motor driver (MC33926 Motor Driver Carrier, Pololu Robotics & Electronics) were

https://doi.org/10.1017/S0263574718000231 Published online by Cambridge University Press

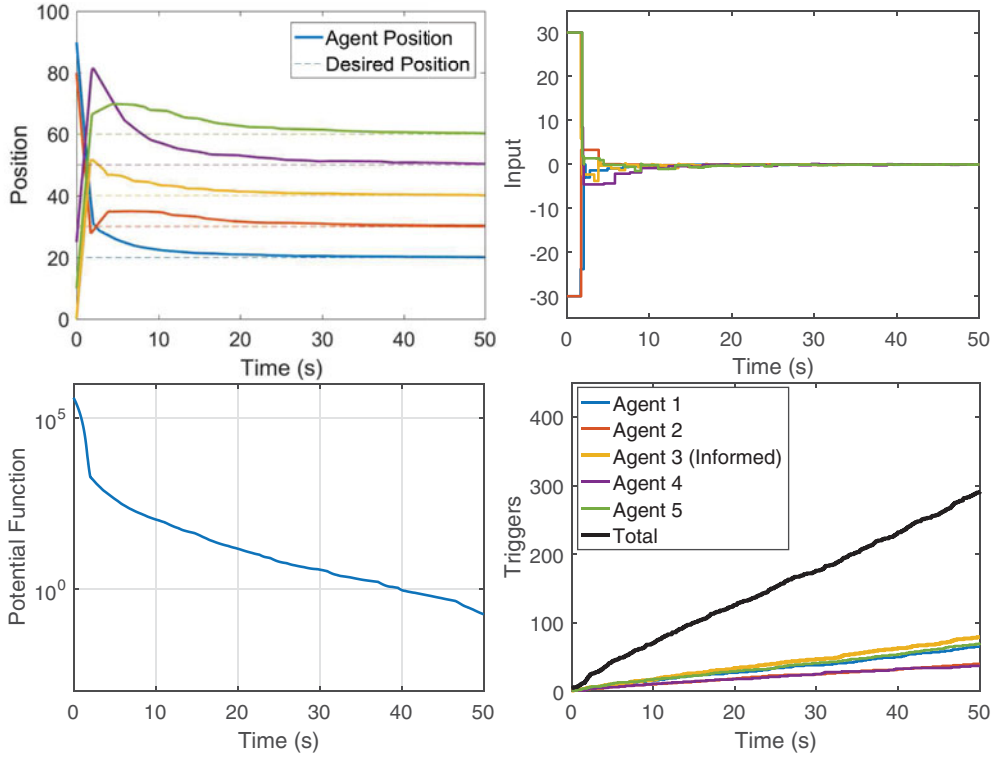

Fig. 3. Simulation of $N = 5$ agents with kinematics (1) and control calculated using Algorithm 1 converging to a desired configuration (top-left). (Top-right) The control plotted vs. time, the potential function (12) plotted vs. time (bottom-left) and the total triggers vs. time (bottom-right).

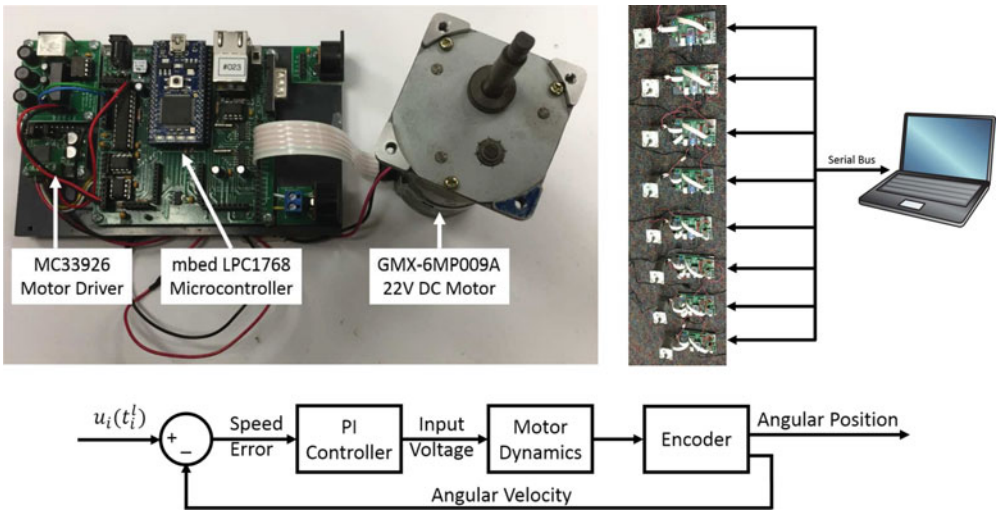

Fig. 4. (Left) A single DC motor modules used to implement Algorithm 1. (Right) Eight motor modules were used in the experiments, each communicates with the central computer (cloud) using a serial interface. (Bottom) Each motor module utilizes an inner loop PI controller to track the desired speed specified by Algorithm 1.

used.[(2)] Figure 4(top-left) shows a single module with important components labeled for reference. Each module is powered by a 20-V DC power supply (PA-1700-02, Safety Mark) resulting in a maximum achievable motor speed of approximately 30 rad/s. Figure 4(top-right) shows a schematic of the eight DC motor modules used during testing. Each module communicates with the cloud (central

[(2)]Extensive documentation of the mbed-based single board computer can be found at: https://developer.mbed.org/users/jebradshaw/code/mbedWSEsbc/wiki/Homepage

https://doi.org/10.1017/S0263574718000231 Published online by Cambridge University Press

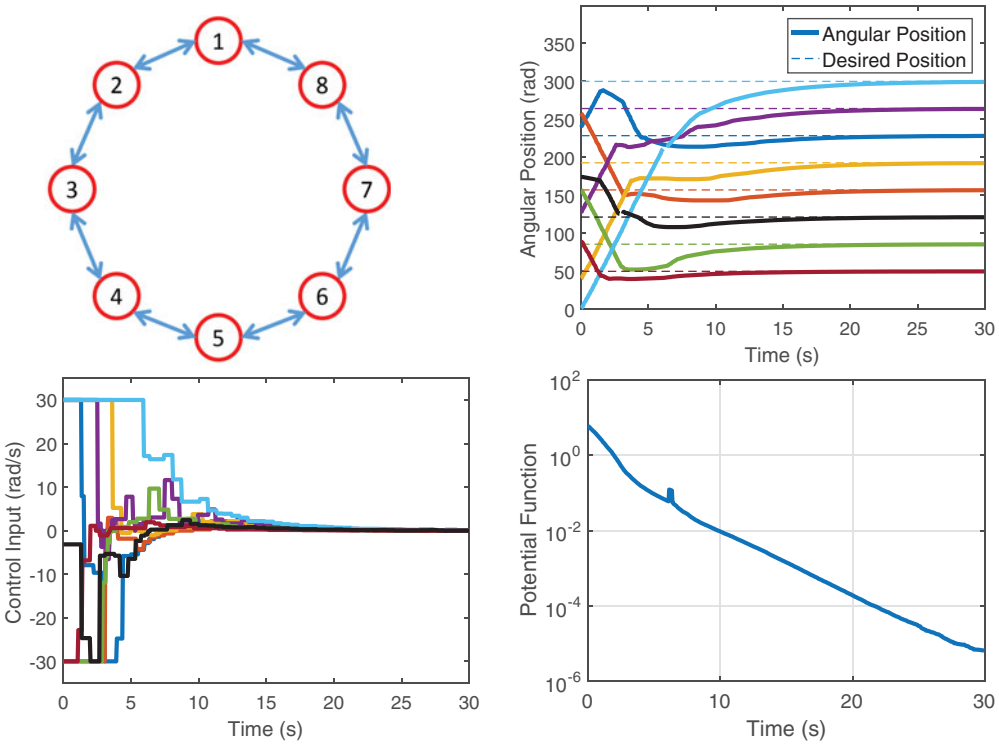

Fig. 5. $N = 8$ DC Motor modules with serial communication to the cloud and implementing control calculated using Algorithm 1. (Top-left) Agent communication topology. (Top-right) Agent positions over time, (bottom-left) control inputs of DC motor modules, and (bottom-right) the potential function (12) plotted vs. time.

computer) through a serial connection. The central computer allows the user to specify the topology of the communication graph and the desired configuration used to generate communication weights.

The mbed microcontroller implements an inner loop proportional-integral (PI) controller tuned to reach a specified speed with settling time of 0.1 s. The PI controller is implemented using Tustin's bilinear approximation[32] with sampling frequency of 100 Hz. As shown in Fig. 4(bottom), the inner loop PI controller receives the desired speed command $u(t_i^l)$ from the distributed self-triggered controller outlined in Algorithm 1. The hierarchical control structure consisting of an inner loop for speed control and a self-triggered outer loop to specify the speed was specifically chosen reminiscent of autonomous underwater,[33] ground,[12] and air[34] vehicle control architectures with inner loop and outer loop characteristics.

For experimental testing and data collection purposes, each module communicates with a central computer over a serial connection. The central computer acts as the cloud server, which in this architecture transmits the control $u_i(t_i^l)$, the interval duration $(t_i^{l+1} - t_i^l)$, and the expiration time within the interval $(t_i^{expire} - t_i^l)$ at triggering times specified by Algorithm 1. Given the maximum speed of the motor, we use a saturation function on the control signal $u_i(t_i^l)$ and promise $M_i(t)$ to remain within the feasible speed range of the motor. The distribution of the control and triggering times is performed in a centralized manner from the main computer for proof of concept, data collection, and post-processing purposes of this paper; full decentralization is the subject of ongoing research. Each module streams its measured angular position in radians, measured speed in rad/s, and applied motor duty cycle for data post-processing purposes. The cloud is updated with an agent's necessary information only at triggering intervals. The dwell time for all experiments is $T_i^{dwell} = 0.4$ s corresponding to four times the settling time of the inner loop PI controller. The self-triggering weighting term was $\sigma = 0.5$ for all agents.

Figure 5 illustrates results of the experimental implementation of Algorithm 1 with $N = 8$ DC motor modules for $t = 30$ s. The modules communicate with the cloud and are given neighbor information according to the cyclic communication topology shown in Fig. 5(top-left). The angular position of each motor module is shown in Fig. 5(top-right). Solid lines illustrate the encoder measured angular

https://doi.org/10.1017/S0263574718000231 Published online by Cambridge University Press

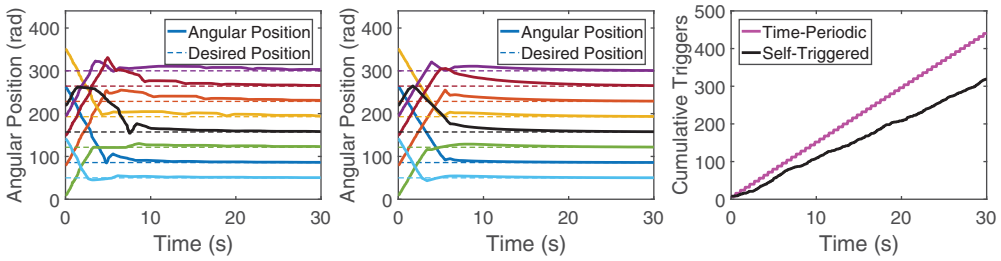

Fig. 6. Experimental results comparing the self-triggered control in Algorithm 1 to a time-periodic implementation triggered every 0.5 s. The self-triggered (left) and time-periodic (middle) responses are similar in structure, but the self-triggered algorithm requires fewer control updates (right).

position in radians, whereas dashed lines represent the desired configuration. Agent one, shown in dark blue, is the only informed agent. All others rely only on inter-agent weights. Note in this experiment two agents suffer dropouts in encoder measurements. Agent eight, illustrated by the black line, loses encoder measurements around $t = 3$ s, and agent six, shown in light blue, loses encoder measurements near $t = 6.3$ s. The agent positions still converge to the desired configuration. Figure 5(bottom-left) shows the control inputs $u_i(t_i^l)$, $i = 1, \ldots, 8$ for all agents over time. Figure 5(bottom-right) shows the potential function (12) over the duration of the experiment, plotted on a log scale. The potential function decreases over the duration of the experiment, with a momentary increase caused by loss of measurements from agent six.

Additional experiments were conducted to compare the performance of Algorithm 1 to a traditional time-periodic implementation. Both the self-triggered and time-periodic algorithms were implemented with the same initial conditions on the motor module network. The cyclic graph communication network shown in Fig. 5(top-left) was used for both experiments. The time-periodic implementation updated each agent control every 0.5 s consistent with the theoretical stability limit defined by $\Delta t_{max} = 2/\max_k(\lambda_k(L))$, where $\lambda_k(L)$ denotes the kth eigenvalue of the graph Laplacian. The interval was found to be quite accurate as longer intervals produced unstable motor responses. The self-triggered algorithm used $\sigma = 0.5$ for all agents consistent with a balanced trade-off of convergence time and number of triggering instances. Both experiments were run for $t = 30$ s.

Figure 6 illustrates a comparison of both experiments. Figure 6(left) shows the angular position of each motor over the duration of the self-triggered experiment, whereas Fig. 6(middle) shows the angular positions during the time-periodic implementation. Note the state trajectories are similar with comparable convergence times. Figure 6(right) compares the cumulative triggering instances of both algorithms. The magenta line shows the number of time-periodic triggers over time. Every agent updates at a triggering event so cumulative number of triggers increases by $N = 8$ at every trigger. The black line shows the cumulative triggers using Algorithm 1. Note that the self-triggered algorithm uses 121 fewer triggering instances than the periodic implementation.

4.2. Simulated application: Quadrotor formation control

The experimental results of Section 4.1 show the utility of the self-triggered controller when applied to a collection of DC motors with inner loop PI controllers. This section utilizes a similar inner- and outer-loop approach to simulate control of a collection of nonlinear quadrotor aircraft to a desired formation.

Figure 7(top) illustrates a block diagram of a single quadrotor model in the multi-quadrotor system. The state of the quadrotor, $\boldsymbol{q} \in \mathbb{R}^{12}$, describes the position (x, y, z), orientation Euler angles (roll, pitch, and yaw), and translational and angular velocities in the body-fixed reference frames (u, v, w), (p, q, r), repsectively. The dynamics of each quadrotor are governed by Newton's second law, derivation of which can be found in ref. [36]. The total force and moment are determined by the sum of thrust forces produced by each motor, which are mapped to four control inputs consisting of the total thrust and moments about each body axis. An attitude controller stabilizes the quadrotor to a desired orientation that minimizes error between the actual and desired total thrust. The desired force is calculated by a velocity controller that compares the desired and actual inertial velocities, where the desired velocity is provided by the self-triggered controller. Note, we assume the attitude and

https://doi.org/10.1017/S0263574718000231 Published online by Cambridge University Press

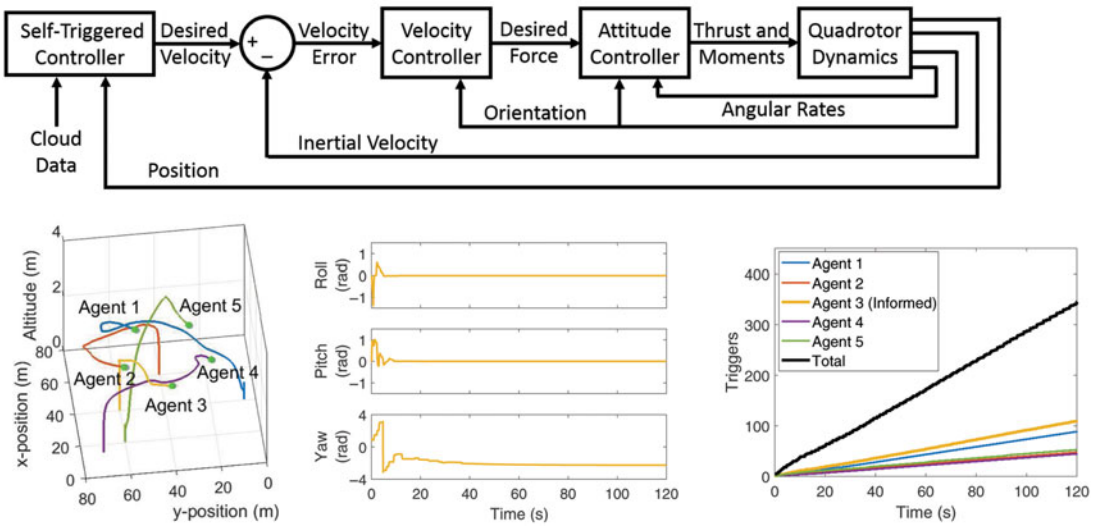

Fig. 7. Simulation of $N = 5$ quadrotors implementing Algorithm 1 to achieve a formation in three dimensions. (Top) block diagram of control implementation, (bottom-left) quadrotor trajectories, (bottom-middle) Euler angles of Agent 3 vs. time, and (bottom-right) cumulative triggers.

velocity controllers are updated continuously, whereas the self-triggered controller, which relies on communication with the cloud, updates only on intervals calculated using Algorithm 1. For derivation and proof of stability of the attitude controller, the interested reader is referred to ref. [36].

The asynchronous, self-triggered control in Algorithm 1 is extended to two dimensions such that the desired velocity vector $\vec{u}(t_i^l) = [u_x(t_i^l), u_y(t_i^l)]^{\mathrm{T}} \in \mathbb{R}^2$ specifies the planar velocity over the time interval $t \in [t_i^l, t_i^{l+1})$. Without loss of generality, the altitude is commanded to a constant value using a proportional altitude controller incorporated into the velocity control block.

Figure 7(bottom) illustrates simulation of $N = 5$ quadrotors commanded to evenly spaced positions about a circle centered at position (30, 40) m with 20 m radius and an altitude of 2.5 m. We assume Agent 3 is the only quadrotor informed of its desired location in the formation, all others must rely on communication with neighbors to achieve their desired position. Agents communicate according to the network topology illustrated in Fig. 2. Figure 7(bottom-left) shows the vehicle trajectories in three dimensions. Gray spheres illustrate the desired configuration positions, whereas green spheres show the achieved vehicle positions at $t = 120$ s. Note that vehicles approach the desired configuration. Figure 7(bottom-middle) shows the Euler angle orientation (roll, pitch, and yaw angles) of quadrotor 3 during the course of the simulation. Note, Agent 3 undergoes a transient phase for approximately 40 s while tracking the desired velocity as it asynchronously updates; after 40 s the quadrotor is in a near hover state as it approaches the desired configuration location. Figure 7(bottom-right) shows the cumulative triggering instances of each agent and the group during the simulation. There are a total of 345 triggering instances, which correspond to the cloud providing information at an average of 2.875 Hz. Total triggering instances vary between agents with average triggering rates between 0.375 Hz and 0.917 Hz.

5. Conclusion

This paper leverages tools from graph theory, Lyapunov-based control, and self-triggered control to derive theoretically justified, distributed control algorithms driving a multi-agent group of agents to a desired configuration. We present a kernel construction and graph transformation technique that invokes properties of the similarity transformation to map the kernel of the Laplacian of an unweighted, connected, undirected graph to the kernel of the Laplacian of a weighted, directed graph with equal topological edges. The transformed graph Laplacian is used to derive a distributed controller in continuous time that steers agents to a desired configuration using the underlying inter-agent communication topology.

https://doi.org/10.1017/S0263574718000231 Published online by Cambridge University Press

Leveraging the derivation of the continuous-time distributed control algorithm and results from self-triggered control,[1,2] we derive a self-triggered controller in which agents asynchronously update their control and communicate with neighbors via a cloud server. The algorithm drives agents to the desired configuration and does not require agents to synchronously update their control at periodic intervals, which decreases the total number of triggering instances.

We validate the theoretically justified algorithms experimentally using a network of eight DC motor modules. Each module uses an mbed microcontroller and a DC motor with an encoder. The microcontroller incorporates an inner-loop PI controller to drive the motor at a desired speed, which is specified by the distributed, self-triggered controller. Results show that the self-triggered algorithm successfully drives the network of agents to a desired configuration of angular positions and decreases communication instances compared to a traditional periodic implementation. Furthermore, simulation of a multi-quadrotor system suggests the self-triggered control approach is a viable option for driving autonomous agents to a desired formation with fewer control update instances required.

Ongoing and future work seeks to extend the graph transformation results to a wider array of graph topologies including directed, weakly connected, and time-varying graphs. These results combined with ongoing control design may enable derivation of self-triggered control approaches capable of steering agents to time-varying configurations while limiting communication between neighbors. We additionally seek to extend the results of this work to higher fidelity models of agent motion including double-integrator dynamics, vehicle-specific models, and heterogeneous agent systems.

Acknowledgments

The authors of this work gratefully acknowledge Joe Bradshaw and Norm Tyson in the United States Naval Academy Technical Support Division for their development of, and assistance with, the DC motor modules used in this work. This work is supported by the Office of Naval Research under grant numbers N0001416WX01249 and N0001415WX01372 and the Trident Scholar Program at the United States Naval Academy.

References

1. C. Nowzari and G. J. Pappas, "Multi-Agent Coordination with Asynchronous Cloud Access," *Proceedings of the American Control Conference*, Boston, MA, (2016) pp. 4649–4654.
2. S. L. Bowman, C. Nowzari and G. J. Pappas, "Coordination of Multi-Agent Systems via Asynchronous Cloud Communication," *Proceedings of the 55th IEEE Conference on Decision and Control*, Las Vegas, NV (IEEE, 2016) pp. 2215–2220.
3. J. S. Wit, Vector Pursuit Path Tracking for Autonomous Ground Vehicles, *Ph.D. Thesis*, (Department of Mechanical Engineering, University of Florida, 2000).
4. C. Nowzari and J. Cortes, "Team-triggered coordination for real-time control of networked cyber-physical systems," *IEEE Trans. Autom. Control* **61**(1), 34–47 (2016).
5. M. H. Degroot, "Reaching a consensus," *J. Am. Stat. Assoc.* **69**(345), 118–121 (1974).
6. X. Qi and Z.-J. Cai, "Three-dimensional formation control based on filter backstepping method for multiple underactuated underwater vehicles," *Robotica* **35**(8), 1690–1711 (2017).
7. M. Tahir and S. K. Mazumder, "Self-triggered communication enabled control of distributed generation in microgrids," *IEEE Trans. Ind. Inform.* **11**(2), 441–449 (2015).
8. Y. Cao, W. Yu, W. Ren, and G. Chen, "An overview of recent progress in the study of distributed multi-agent coordination," *IEEE Trans. Ind. Inform.* **9**(1), 427–438 (2013).
9. J. Baikerikar, S. Surve and S. Prabhu, "Consensus Based Dynamic Load Balancing for a Network of Heterogeneous Workstations," **In**: *Advances in Computing, Communication and Control* (S. Unnikrishnan, S. Surve and D. Bhoir, eds.) (Springer, Berlin, Heidelberg, 2011) pp. 116–124.
10. W. Ren, "Multi-vehicle consensus with a time-varying reference state," *Syst. Control Lett.***56**, 474–483 (2007).
11. A. Chapman, *Advection on Graphs* (Springer International Publishing, Cham, 2015) pp. 3–16.
12. R. Falconi, L. Sabattini, C. Secchi, C. Fantuzzi and C. Melchiorri, "Edge-weighted consensus-based formation control strategy with collision avoidance," *Robotica* **33**(2), 332–347 (2015).
13. W. L. Seng, J. C. Barca and Y. A. Şekercioğlu, "Distributed formation control of networked mobile robots in environments with obstacles," *Robotica* **34**(34), 1403–1415 (2016).
14. W. Heemels, K. Johansson and P. Tabuada, "An Introduction to Event-Triggered and Self-Triggered Control," *Proceedings of the 51st Conference on Decision and Control*, Maui, HI (2012) pp. 3270–3285.
15. M. Mesbahi and M. Egerstedt, *Graph Theoretic Methods in Multiagent Networks* (Princeton University Press, Princeton, New Jersey, USA, 2010).

16. R. Olfati-Saber, J. A. Fax and R. M. Murray, "Consensus and cooperation in networked multi-agent systems," *Proc. IEEE* **95**(1), 215–233 (Jan. 2007).
17. J. A. Fax and R. M. Murray, "Information flow and cooperative control of vehicle formations," *IEEE Trans. Autom. Control* **49**(9), 1465–1476 (Sep. 2004).
18. W. Ren and R. Beard, *Distributed Consensus in Multivehicle Cooperative Control: Theory and Applications* (Springer-Verlag, London, 2008).
19. M. H. Yamchi and R. M. Esfanjani, "Formation control of networked mobile robots with guaranteed obstacle and collision avoidance," *Robotica* **35**(6), 1365–1377 (Jun. 2017).
20. D. V. Dimarogonas, E. Frazzoli and K. H. Johansson, "Distributed event-triggered control for multi-agent systems," *IEEE Trans. Autom. Control* **57**(5), 1291–1297 (2012).
21. P. Tabuada, "Event-triggered real-time scheduling of stabilizing control tasks," *IEEE Trans. Autom. Control* **52**(9), 1680–1685 (2007).
22. J. Araújo, A. Anta, M. Mazo, J. Faria, A. Hernandez, P. Tabuada and K. H. Johansson, "Self-Triggered Control Over Wireless Sensor and Actuator Networks," *Proceedings of the International Conference on Distributed Computing in Sensor Systems and Workshops DCOSS*, Barcelona, Spain (2011) pp. 1–9.
23. T. Liu and Z. P. Jiang, "A small-gain approach to robust event-triggered control of nonlinear systems," *IEEE Trans. Autom. Control* **60**(8), 2072–2085 (2015).
24. P. V. Teixeira, D. V. Dimarogonas, K. H. Johansson and J. Sousa, "Event-Based Motion Coordination of Multiple Underwater Vehicles Under Disturbances," *Proceedings of the OCEANS Conference OCEANS 2010*, Sydney, Australia (2010) pp. 1–6.
25. A. Adaldo, D. Liuzza, D. V. Dimarogonas and K. H. Johansson, "Control of Multi-Agent Systems with Event-Triggered Cloud Access," *Proceedings of the European Control Conference*, Linz, Austria (2015) pp. 954–961.
26. A. Bondy and U. Murty, *Graph Theory*, 1st ed. (Springer-Verlag, London, 2008).
27. L. Moreau, "Stability of multiagent systems with time-dependent communication links," *IEEE Trans. Autom. Control* **50**(2), 169–182 (2005).
28. R. Horn, *Matrix Analysis* (Cambridge University Press, New York, NY, 1985).
29. L. DeVries and M. D. M. Kutzer, "Kernel Design for Coordination of Autonomous, Time-Varying Multi-Agent Configurations," *Proceedings of the American Control Conference*, Boston, MA (2016) pp. 1975–1980.
30. R. Sepulchre, D. A. Paley and N. E. Leonard, "Stabilization of planar collective motion: All-to-all communication," *IEEE Trans. Autom. Control* **52**, 811–824 (May 2007).
31. H. K. Khalil, *Nonlinear Systems*, 3rd ed. (Prentice Hall, Upper Saddle River, New Jersey, USA, 2002).
32. N. Nise, *Control Systems Engineering*, 7th ed. (Wiley & Sons, Hoboken, NJ, 2015).
33. C. Xinghua and L. Juan, "AUV Planner Tracking Control Based on the Line of Sight Guidance Method," *Proceedings of the IEEE Conference on Mechatronics and Automation*, Tianjin, China (2014) pp. 1204–1208.
34. M. C. P. Santos, M. Sarcinelli-Filho and R. Carelli, "Trajectory Tracking for UAV with Saturation of Velocities," *Proceedings of the International Conference on Unmanned Aircraft Systems ICUAS*, Arlington, VA (2016) pp. 643–648.
35. A. Meenakshi and C. Rajian, "On a product of positive semidefinite matrices," *Linear Algebra Appl.* **295** (1-3), 3–6 (1999).
36. D. Mellinger, Trajectory Generation and Control for Quadrotors *Ph.D. Dissertation* (Department of Mechanical Engineering and Applied Mechanics, University of Pennsylvania, 2012).
37. B. Acikmese, "Spectrum of Laplacians for graphs with self-loops," ArXiv:1505.08133 [math.OC] (2015).
38. B. Acikmese, M. Mandić and J. L. Speyer, "Decentralized observers with consensus filters for distributed discrete-time linear systems," *Automatica* **50**(4), 1037–1052 (2014).

Appendix

Proof of positive semi-definiteness of $R^{-1}L^2R^{-1}$

Lemma 2. *Given the diagonal positive definite matrix R defined by (4) and the Laplacian L of a strongly connected, undirected graph,* $\mathcal{G}$*, the quantity* $R^{-1}L^2R^{-1}$ *is positive semi-definite and symmetric.*

Proof. The graph Laplacian is a symmetric, positive semi-definite matrix for strongly connected, undirected graphs.[26] R and therefore, R^{-1} are symmetric, positive definite matrices by construction. $R^{-1}L^2R^{-1}$ is positive semi-definite since the product of two symmetric positive semi-definite (definite) matrices are positive semi-definite (definite) if and only if the product is normal.[35] Normality can be shown since R and L are symmetric. □

https://doi.org/10.1017/S0263574718000231 Published online by Cambridge University Press

Proof of positive definiteness of $W = R^{-1}(L + \Lambda)R^{-1}$

Lemma 3. *Given the diagonal positive definite matrix R defined by (4), the Laplacian L of a connected, undirected graph,* $\mathcal{G}$*, and the informed agent matrix* Λ*, the quantity* $W = R^{-1}(L + \Lambda)R^{-1}$ *is symmetric and positive definite if at least one agent is informed.*

Proof. The quantity $L + \Lambda$ equivalently represents an undirected, connected graph with self-loops, where a self-loop is defined as an agent with a communication edge with itself. The Laplacian of a connected graph with self-loops is positive definite;[37,38] therefore, $L + \Lambda$ is positive definite.

Since R^{-1} is positive definite and diagonal and $L + \Lambda$ is positive definite and symmetric, the product $R^{-1}(L + \Lambda)$ is positive definite because it is normal.[35] By the same logic, $R^{-1}(L + \Lambda)R^{-1}$ is positive definite since each term is symmetric and positive definite.[35] Symmetry of W follows since each matrix composing it is symmetric. □

Positive definiteness of Q

Lemma 4. *Given the diagonal positive definite matrix R defined by (4), the Laplacian L of a strongly connected, undirected graph,* $\mathcal{G}$*, and the informed agent matrix* Λ*, the quantity* $Q = R^{-1}(L + \Lambda)^2R^{-1}$ *is positive definite if at least one agent is informed.*

Proof. The proof of Lemma 4 follows from Lemmas 2 and 3 and is omitted for brevity. □

https://doi.org/10.1017/S0263574718000231 Published online by Cambridge University Press