Automatic Generation of Persistent Formations for Multi-Agent Networks Under Range Constraints

Brian Smith, Magnus Egerstedt, and Ayanna Howard

***Abstract*—We present graph-based methods for determining if a mobile robot network with a defined sensor and communication range can *persistently* achieve a specified formation, which implies that the formation, once achieved, will be preserved by the direct maintenance of a subset of inter-agent distances. Here, formations are defined by a set of points whose inter-point distances correspond to desired inter-agent distances. Further, we provide graph operations to describe agent interactions that implement a given formation, as well as an algorithm that, given a persistent formation, automatically generates a sequence of such operations.**

I. Introduction

Due to recent developments in mobile sensing, computation, and actuation, formation control for multi-agent networks has received much attention. For example, see [1], [2], [3], [4], [5], and [6]. Recent research suggests the use of graph-theoretic structures to represent formations, where vertices represent agents, and edges represent specific inter-agent distances to be maintained through decentralized control laws [7], [8], [9].

In this paper, we will study such graph-based abstractions of formations, and we define a *target formation* as a set of desired inter-agent distances associated with a complete graph, i.e., these distances are specified with respect to all pairs of agents. However, it may not be the case that all inter-agent distances are needed, which leads to the study of so-called *persistent formations* [10]. In a persistent formation, each agent is assigned a set of *constraints*, which are specific inter-agent distances to maintain. These constraints are oriented in the sense that each constraint is the responsibility of a single agent rather than two agents. Moreover, since not all inter-agent distances are explicitly needed in order to preserve the formation, persistent formations typically only involve a proper subset of all possible inter-agent constraints. In [11], graph operations are proposed that, through successive applications, produce a graph corresponding to a persistent formation.

We want to use such operations in order to build persistent formations in the presence of constraints on the effective communication and sensing distances. Unfortunately, these types of constraints are not considered in [11], and the main contribution of this paper is the sequential construction of persistent formations that respect the inter-agent range constraints. As a consequence, we produce a method for determining if a specific target formation is *persistently feasible* with respect to the range constraints.

Brian Smith, Magnus Egerstedt, and Ayanna Howard are with the School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA, USA. Email: {brian, magnus, ayanna.howard}@ece.gatech.edu

II. Preliminaries

In this section, we review some of the basic assumptions and terminology needed for the development in later sections. The main object of interest is that a persistent formation in which individual robots are responsible for maintaining specific inter-agent distances. Qualitatively, we say that a formation is persistent if, provided that all agents ensure that the distance constraints they are responsible for are satisfied, then the formation is preserved [10].

There are two problems addressed in this paper: (1) determine if a target formation is persistently feasible given the maximum sensing and communication range of the agents; and, (2) if the formation is persistently feasible, generate a graph that represents the target formation. We assume that the desired inter-agent distances can be represented by a given set of n points $P = \{p_i : p_i \in \mathbb{R}^2\}$, where the distances between each pair of points describes the desired distances between each pair of agents. This set of points defines a *target formation*. We are also given a *proximity range* $\Delta \in \mathbb{R}$, defining the range at which agents can sense and communicate with each other. Any pair of agents i and j can sense and communicate with each other if and only if their distance is less than Δ, and that agents i and j can *directly* maintain their current distance only if they are within Δ.

III. Frameworks and Rigidity

In this section, we present the concepts of frameworks and rigidity.

A. Frameworks

Given an initial set of n points P, we define a *framework* [12] as $G(P) = (P, G_n)$, where $G_n = (V, E)$ is a graph such that

- $V = \{1, \ldots, n\}$ is the vertex set with $i \in V$ is the vertex corresponding to $p_i \in P$, and
- $E \subset V \times V$ is an edge set, where each edge $e \in E$ is a pair of vertices (i, j) such that $i \neq j$.

We also define a weight function $\delta : E \rightarrow \mathbb{R}$ which assigns to each edge the *distance* between the points incident to that edge (i.e., $\delta(i, j) = \|p_i - p_j\|$). Frameworks are used to represent both the target formation (*configuration* P) and the inter-agent distances that will be directly maintained (graph G_n).

B. Rigidity

As in [7], we define a *trajectory* of a framework $G(P)$ as a set of n states $X(t) = \{x_i(t) : x_i(0) = p_i(0), t \geq 0\}$ such that each state is continuous. A trajectory represents the *motion* of a multi-agent network that is initially in the desired target formation, and we define a *edge-consistent trajectory* as one such that $\|x_i(t) - x_j(t)\|$ is constant for all $(i, j) \in E$. We define a *rigid trajectory* as one such that the distances between every pair of states $x_i(t)$ and $x_j(t)$ remain constant. Thus, a rigid trajectory represents a *rigid motion* of the network, starting from the target formation, during which all inter-agent distances are maintained. A framework is *rigid* if and only if all edge-consistent trajectories of the framework are rigid trajectories. If a framework is not rigid, we say that it is *flexible*. Rigid frameworks represent *rigid formations*. Their existence for a particular configuration P implies that the target formation can be maintained by guaranteeing that the inter-agent distances corresponding to the edges of the framework are maintained. Figure 1 gives examples of rigid and flexible frameworks.

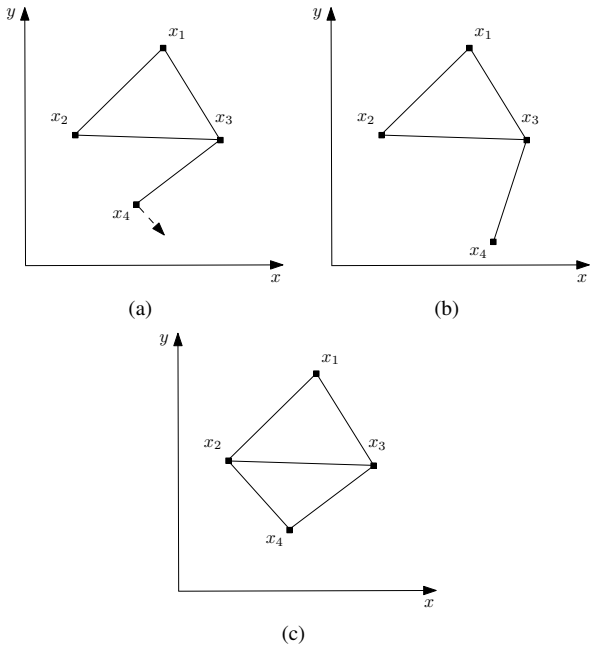

Fig. 1. Rigid and Flexible Frameworks. 1(a): A flexible framework. The dotted line represents the direction of circular motion that agent 4 can take and still satisfy its constraint with agent 3. 1(b): Agent 4 can move in a manner that changes its distance to agents 1 and 2. 1(c): A rigid framework. If all agents satisfy their constraints, the formation does not change.

IV. Directed Frameworks and Persistence

A. Directed Frameworks

Here, we assume that, for each pair of agents whose distance is directly maintained, the responsibility of its maintenance is delegated to a single agent of the pair. We represent this by adding a direction to each edge. We represent a *directed framework* by $\vec{G}(P) = (P, \vec{G}_n)$, where $\vec{G}_n = (V, \vec{E})$ is a *directed graph*. Here, $(i, j) \in \vec{E}$ indicates a directed edge from i to j. The presence of such an edge indicates that the control laws of agent i should maintain the distance of edge (i, j). We call this a *constraint* of agent i.

B. Persistence

Persistence is a quality of directed frameworks, and is very closely related to the concept of *constraint consistence*. Informally, we say that constraint consistence means that all constraints are satisfied as long

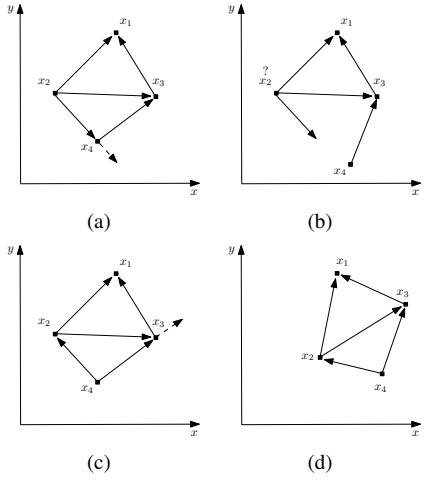

Fig. 2. Persistence and constraint consistence. 2(a): A framework that is not persistent. It is rigid, but not constraint consistent. 2(b): If agent 4 moves, agent 2 cannot move in a way that preserves the distances between agent 2 and agents 1, 3, and 4. 2(c): A persistent framework. It is constraint consistent and rigid. Agents 1 and 3 are a leader-follower pair. 2(d): If agent 3 moves, all agents can still satisfy their constraints.

as all agents satisfy their individual constraints, i.e., no subset of agents can satisfy their constraints in a manner which forces another agent to not satisfy a constraint. Constraint consistence is, to a large degree, determined by the number and orientation of the framework edges. Figure 2 shows constraint consistent and inconsistent frameworks. For a more rigorous definition, see [10].

The following theorem describes persistence:

Theorem 1. *[10] A framework is persistent if and only if it is rigid and constraint consistent.*

Persistent frameworks represent *persistent formations*. Their existence for a particular configuration P implies that the target formation can be maintained like a rigid formation in a constraint consistent manner. Figures 2(c) and 2(d) shows a persistent framework.

V. Persistent Feasibility and Persistent Graph Generation: The Modified "Pebble Game"

In this section, we present a method for determining if a target formation is persistently feasible. We define the concepts of infinitesimal and generic rigidity, as well as rigid and persistent feasibility. An algorithm for determining rigid feasibility is presented. We then demonstrate that rigid feasibility and persistent feasibility are equivalent. We also show that the same algorithm generates minimally persistent graphs.

A. Infinitesimal Rigidity

The concept of *infinitesimal rigidity* is discussed in [13], [14], and [15]. We assume that $x_i(t)$ is a differentiable function for each vertex i in the framework. Since we have defined an edge-consistent trajectory of $G(P)$ such that the distance between points $x_i(t)$ and $x_j(t)$ remains constant all along the trajectory (i.e., $\|x_i(t) - x_j(t)\|^2 = constant$ for all $(i,j) \in E$, this implies that

$$(\dot{x}_i(t) - \dot{x}_j(t)) \cdot (x_i(t) - x_j(t)) = 0 \ \forall (i,j) \in E. \quad (1)$$

The assignment of constant instantaneous velocities $\dot{x}_i = u_i$ that satisfy Equation 1 at $t = 0$ is described as an *infinitesimal motion* of the framework [15]. Let $\dot{u}$ be the point in $\mathbb{R}^{n2}$ defined by the infinitesimal motion such that $\dot{u} = (\dot{u}_1^T, \dot{u}_2^T, \ldots, \dot{u}_n^T)^T$. Equation 1 can thus be represented in matrix form as

$$M(G(P))\dot{u} = 0,$$

where $M(G(P))$ is known as the *rigidity matrix* [15]. The rigidity matrix has $card(E)$ rows and $2n$ columns, where "card" denotes cardinality. For each edge (i,j), Each row m_{ij} of $M(G(P))$ represents the equation for that edge as

$$m_{ij} = (0, \ldots, (p_i - p_j)^T, 0, \ldots, 0, (p_j - p_i)^T, \ldots, 0).$$

Here, $(p_i - p_j)^T$ in two columns for vertex i, $(p_j - p_i)^T$ in the columns for j, and zero elsewhere [15]. A framework with $n \geq 2$ points in $\mathbb{R}^2$ is *infinitesimally rigid* if and only if $rank(M(G(P))) = 2n - 3$ [14], [15] .

The following theorem establishes the relationship between infinitesimal rigidity and rigidity:

Theorem 2. *[13] Infinitesimal rigidity implies rigidity.*

Note, however, that rigidity does not imply infinitesimal rigidity. For a more detailed description, see [13], [14], and [15].

B. Generic Rigidity

It is clear that the rigidity of a framework $G(P)$ depends both on the topology and the configuration. For a given graph G_n, we can think of a framework $G(P) = (P, G_n)$ as a *realization* of G_n, and we define a *generically rigid graph* as follows:

Definition 1. *A graph is generically rigid if it has an infinitesimally rigid realization.*

Note that generic rigidity is a property of a graph, not a framework. Therefore, we refer to generically rigid graphs as *rigid graphs* without confusion. If G_n is a rigid graph, and $G(P) = (P, G_n)$ is infinitesimally rigid, we say that P is a *generic configuration* for G_n, and that $G(P)$ is a *generic realization*.

The configuration $P = \{p_1, \ldots, p_n\}$ defines a point in $\mathbb{R}^{n2}$ as $p = (p_1, \ldots, p_n)^T$. The following lemma describes the set of generic configurations for a generically rigid graph:

Lemma 1. *[14] If G_n is a generically rigid graph, then the generic configurations for G_n form a dense, open subset of $\mathbb{R}^{n2}$.*

This implies that, for any generically rigid graph G_n, any configuration $P' = \{p'_1, \ldots, p'_n\}$ that defines a point $p' \in \mathbb{R}^{n2}$ as $p = (p'_1, \ldots, p'_n)^T$ can be well-approximated by a generic configuration $P = \{p_1, \ldots, p_n\}$ that defines a point $p \in \mathbb{R}^{n2}$ as $p = (p_1, \ldots, p_n)^T$ such that $G(P) = (P, G_n)$ is infinitesimally rigid and, therefore, rigid by Theorem 2.

C. Rigid Feasibility

Rigid feasibility is defined as follows:

Definition 2. *A target formation defined by the configuration P is* rigidly feasible *for a multi-agent network*

with proximity range Δ if and only if there exists a framework $G^{\Delta}(P) = (P, G_n^{\Delta})$ such that $G_n^{\Delta} = (V, E^{\Delta})$ is rigid, and $\delta(e) \leq \Delta \ \forall e \in E^{\Delta}$.

It is clear that adding edges to a rigid graph cannot affect its rigidity. We define a *minimally rigid graph* as follows:

Definition 3. *A graph is* minimally rigid *if and only if it is rigid but does not remain rigid after the removal of a single edge.*

The following theorem gives necessary and sufficient conditions for a graph to be minimally rigid:

Theorem 3. *[16] A graph with $n \geq 2$ vertices in $\mathbb{R}^2$ is minimally rigid if and only if*

1) *it has $2n-3$ edges and*
2) *each induced subgraph of $n' \leq n$ vertices has no more than $2n'-3$ edges.*

To generate minimally rigid graphs, we utilize the *"pebble game" algorithm* [17]. It is a proven algorithm for constructing minimally rigid graphs, with a worst case performance of $O(n^2)$ [17].

In the pebble game, each vertex is represented as having two pebbles, each pebble representing a degree of freedom for that vertex. A *pebble covering* exists if each edges can be covered by a pebble from a vertex incident to that edge so that no subgraph has more than $2n-3$ edges. To keep track of pebbles, the pebble game works with a directed graph $\vec{G}_n$, where a directed edge (i,j) indicates that edge (i,j) is covered by a pebble from vertex i. The pebble game starts with a directed graph $\vec{G}_n$ with no edges and attempts to add each potential edge one at a time to the pebble covering, verifying that part 2 of Theorem 3 is satisfied. Since the pebbles of each vertex limit the number of edges directed out of each vertex, this is accomplished by modifying the directions of both the edge to be added and the other edges already in $\vec{G}_n$. If $2n-3$ such edges are added to the graph, then this implies that part 1 of Theorem 3 is satisfied, implying that $\vec{G}_n$ is minimally rigid. For more detail on the implementation of this algorithm, see [17].

To test for a minimally rigid graph that satisfies Definition 2, we modify the pebble game algorithm so that it only considers edges that are less than or equal to Δ. The modified pebble game is described in Algorithm 1.

The following theorem states the effectiveness of the modified pebble game to test for rigid feasibility:

Theorem 4. *A target formation defined by configuration P is rigidly feasible for a multi-agent network with proximity range Δ if and only if the algorithm $ModifiedPebbleGame(P, \Delta)$ returns a minimally rigid graph.*

Algorithm 1 $ModifiedPebbleGame(P, \Delta)$

Require: P is a configuration of n points
  Initialize $\vec{G}_n^{\Delta} = (V, \vec{E}^{\Delta})$ such that $V = \{1, \ldots, n\}, \vec{E} = \emptyset$
  for all possible edges $e = (i,j)$ such that $\delta(e) \leq \Delta$ and **while** $\|\vec{E}^{\Delta}\| < 2n-3$ **do**
    Add edge e to $\vec{E}^{\Delta}$ and rearrange existing edge directions to find a pebble covering;
    if a valid pebble covering is not found **then**
      Remove edge e from $\vec{E}^{\Delta}$;
    else
      Keep edge e in $\vec{E}^{\Delta}$;
    end if
  end for
  if $\|\vec{E}^{\Delta}\| = 2n-3$ **then**
    $rigid$ =**true**;
  else
    $rigid$ =**false**;
  end if
  return [$rigid$, $\vec{G}_n^{\Delta}$];

Proof: Definition 2 is satisfied for configuration P only if there exists a rigid graph G_n^{Δ} such that all edges of the framework $G^{\Delta}(P) = (P, G_n^{\Delta})$ have edges of less than or equal to Δ. By Definition 3, this implies the existence of a minimally rigid graph with the same properties. Assume that $G^{\Delta}(P)$ exists, but that $ModifiedPebbleGame(P, \Delta)$ fails to return a minimally rigid graph. Note from [17] that the unmodified pebble game generates a rigid graph by considering each edge and adding it to a flexible graph until it becomes minimally rigid. Therefore, the failure of $ModifiedPebbleGame(P, \Delta)$ implies that no such graph can be generated considering only edges such that their weight in a framework would be less than or equal to Δ. Since the unmodified pebble game always returns a minimally rigid graph, this implies that all rigid graphs result in a framework with an edge greater than Δ. However, this violates our assumption that $G^{\Delta}(P)$ exists. Therefore, by Definition 2, the formation is rigidly feasible only if $ModfiedPebbleGame(P, \Delta)$ returns a minimally rigid graph.

If the modified pebble game does produce such a minimally rigid graph such that all edges of the framework $G^{\Delta}(P)$ have lengths less than or equal to Δ, then the conditions of Definition 2 are satisfied. ■

D. Persistent Feasibility

Similar to generic rigidity, we say that a graph is *generically persistent* if it has an infinitesimally rigid realization that is persistent. Like generic rigidity, generic persistence applies to graphs, not frameworks. Therefore, we refer to generically persistent graphs as persistent graphs without confusion. A persistent graph is *minimally persistent* if it is persistent and if no edge can be removed without losing persistence [10].

Persistent feasibility is defined as follows:

Definition 4. *A target formation defined by configuration P is* persistently feasible *for a multi-agent network with proximity range Δ if and only if a exists a framework $\vec{G}^{\Delta}(P) = (P, \vec{G}_n^{\Delta})$ such that $\vec{G}_n^{\Delta} = (V, \vec{E}^{\Delta})$ is persistent, and $\delta(e) \leq \Delta \forall e \in \vec{E}^{\Delta}$.*

The following theorem states a minimally persistent graph can be obtained from any minimally rigid graph:

Theorem 5. *[11] For any minimally rigid graph, it is possible to assign directions to the edges such that the obtained directed graph is minimally persistent.*

The following theorem describes the necessary and sufficient conditions for a target formation to be persistently feasible:

Theorem 6. *For a multi-agent network with proximity range Δ, a configuration P defines a persistently feasible target formation if and only if it is rigidly feasible.*

Proof: If P is rigidly feasible, then, by Definitions 2 and 3, there exists a minimally rigid graph G_n^{Δ} such that $G^{\Delta}(P) = (P, G_n^{\Delta})$ is rigid and, for all edges e, $\delta(e) \leq \Delta$. By Theorem 5, this implies the existence of a persistent graph $\vec{G}_n$, and thus a persistent framework $\vec{G}^{\Delta}(P) = (P, \vec{G}_n)$ that satisfies Definition 4. If P is not rigidly feasible, then Theorem 1 implies that P is not persistently feasible. ■

Therefore, the modified pebble game tests for both rigid and persistent feasibility.

E. Persistent Graph Generation

Here, we show that the pebble game algorithm also generates minimally persistent graphs, and thus a persistent framework to represent a persistent formation.

The following theorem gives necessary and sufficient conditions for minimum persistence:

Theorem 7. *[10] A graph is minimally persistent if and only if it is minimally rigid and no vertex has an out-degree larger than 2.*

We denote the out-degree of vertex i by $d^-(i)$. Note that the pebble game produces a directed graph $\vec{G}_n$, where each edge (i, j) is covered by one of two pebbles from vertex i. Thus, we have the following theorem:

Theorem 8. *The pebble game algorithm generates minimally persistent graphs.*

Proof: Assume that $\vec{G}_n$ is a rigid graph successfully generated by the pebble game. [17] already shows that the pebble game generates a minimally rigid graph. Since each directed edge (i, j) represents the edge being covered by a pebble from vertex i, this implies that each vertex i, $d^-(i) \leq 2$. Therefore, Theorem 7 implies that $\vec{G}_n$ is minimally persistent. ■

VI. Graph Operations

In this section, we describe methods for representing and choosing leader-follower pairs of a persistent formation. We present graph operations that represent agent interactions that execute a persistent formation. We also present an algorithm for generating a sequence of graph operations, which represents a sequence of agent interactions to execute a persistent formation.

We define a *leader-follower pair* [7] as a pair of adjacent vertices $i, j \in V$ such that vertex i has an out-degree of 0 and vertex j has an out-degree of 1, and $\exists (j, i) \in E$. We say that vertex i is the *leader vertex*, and vertex j is the *follower vertex*. The leader agent has no constraints, and thus has two degrees of freedom, implying that the persistent formation will follow the leader agent in $\mathbb{R}^2$. Similarly, the follower agent has one constraint, and thus one degree of freedom, implying that the persistent formation will rotate around the leader agent as the follower agent performs circular motion around the leader. For a persistent formation, edge-reversing operations can make any pair of adjacent agents a leader-follower pair [11]. A leader-follower pair is demonstrated in Figures 2(c) and 2(d).

A. Persistent Graph Operations

In an actual multi-agent network, achieving a persistent formation requires agents with no constraints to interact and establish constraints. Such a sequence of agent interactions, if successful, would result in a persistent formation, with inter-agent distances corresponding to the target formation.

Graph operations can be used to represent such a sequence of agent interactions. [11] presents graph operations for assembling and modifying persistent graphs. These operations consist of directed vertex addition and edge-splitting operations. Graph operation sequences are typically generated by performing inverse graph operations on the graph to be assembled, along with edge reversing operations.

[11] shows that any persistent graph can be deconstructed by a combination of these inverse operations, and then reconstructed by a reverse sequence of non-inverse operations. Additionally, [11] guarantees that each intermediate graph is persistent. However, these methods are completely graph based, and do not take into account a proximity range for a multi-agent network. Consider Figure 4(d). This framework has a minimally rigid graph. An inverse vertex addition cannot be performed. Also, note that any inverse edge-splitting operation will introduce a new edge into the framework which has a length longer than any other edge. If Δ had been defined to be equal to the maximum edge length of the longest original edge, this new edge would violate the proximity range of the mobile agent network. *Therefore, given a proximity range limit on the edge lengths of a framework, certain configurations cannot be deconstructed by these traditional operations without introducing an edge that violates the proximity range.*

B. Persistent-Δ Operations

In this section we present two new graph operations to construct persistent graphs. These, combined with traditional vertex addition, allow any persistent graph with a leader-follower pair to be constructed without using any edges that are not contained in the final graph. We call this set of three graph operations *persistent-Δ operations*.

Each operation is represented by a double $op = (V, E)$, where V is a set of vertices to add to the graph, and E is a set of edges to add to the graph.

Consider a directed graph $\vec{G}_n = (V, \vec{E})$ such that $j, k \in V$. *Vertex addition* consists of adding a vertex i to V and adding edges $(i, j), (i, k)$ to $\vec{E}$. A vertex addition is represented as $vertexAddition(i, j, k) = (\{i\}, \{(i, j), (i, k)\})$.

Consider a directed graph $\vec{G}_n = (V, \vec{E})$ such that $j \in V$. *Single-vertex addition* consists of adding a vertex i to V and adding edge (i, j) to $\vec{E}$. A single-vertex addition is represented as $singleVertex(i, j) = (\{i\}, \{(i, j)\})$. Note that this operation does *not* preserve persistence. In fact, it guarantees a loss of persistence, since this new vertex has one degree of freedom.

Consider a directed graph $\vec{G}_n = (V, \vec{E})$ such that $i, j \in V$ and $(i, j) \notin \vec{E}$. *Edge insertion* consists of adding edge (i, j) to $\vec{E}$. An edge insertion is represented as $edgeInsertion(i, j) = (\emptyset, \{(i, j)\})$. Figure 3 shows these operations.

C. Persistent-Δ Sequence Generation

This section describes how persistent-Δ operations can be used to construct any persistent graph with a leader-follower pair.

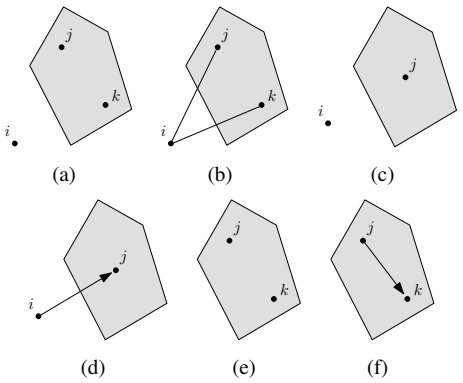

Fig. 3. Persistent-Δ graph operations. 3(a), 3(b): a vertex addition. 3(c), 3(d): a single-vertex addition. 3(e), 3(f): an edge insertion operation.

The following lemma is used to design an effective sequence of persistent-Δ operations and for proving its effectiveness:

Lemma 2. *[11] Let $\vec{G}_n$ be a minimally persistent graph such that vertex $d^-(i) \geq 1$ and vertex $d^-(j) \leq 1$. Then, there is a directed path from i to j.*

We also use the following theorem:

Theorem 9. *[10] Let $\vec{G}_n$ be a minimally persistent graph such that $i, j \in V$ are a leader-follower pair. This implies that, for all vertices $k \in V \setminus \{i, j\}$, $d^-(k) = 2$.*

This leads to the following lemma:

Lemma 3. *Let $\vec{G}_n = (V, \vec{E})$ be a minimally persistent graph such that vertex i is the leader and vertex j is the follower of a leader-follower pair. This implies the existence of a directed path from all vertices $k \in V \setminus i$ to i.*

Proof: Assume that $\vec{G}_n$ exists as in Lemma 3. Since i and j are a leader-follower pair, this implies a directed path from j to i and that $d^-(i) < d^-(j) \leq 1$. By Theorem 9, all vertices $k \in V \setminus \{i, j\}$, $d^-(k) = 2$. By Lemma 2, there exists path from all vertices in $V \setminus i$ to i. ■

This leads us to an algorithm for constructing a sequence of graph operations to construct a minimally persistent graph. We define a *leader-follower seed* as a graph $\vec{G}_0 = (V_0, \vec{E}_0)$ such that $V_0 = \{i, j\}$ and $\vec{E}_0 = \{(j, i)\}$. Here, vertex i is the leader vertex, and vertex j is the follower vertex.

Any minimally persistent graph can be constructed from a leader-follower seed by a sequence of persistent-Δ graph operations. First, given a minimally persistent

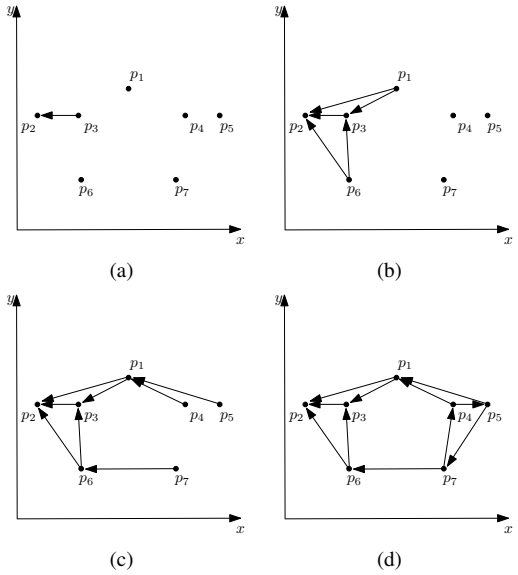

Fig. 4. A sequence of Persistent-Δ operations constructing a framework. 4(a): The initial leader-follower seed. 4(b): Two vertex additions are performed. 4(c): No more vertex additions are possible. Three single-vertex additions are performed. 4(d): Three edge insertions are performed, one for each single-vertex addition.

graph $\vec{G}_n$, a leader-follower seed $\vec{G}_0$ is initialized using the leader and follower vertices in $\vec{G}_n$. Until all vertices and edges of $\vec{G}_n$ are present in $\vec{G}_0$, the following process is performed:

1) Generate each possible edge insertion.
2) Generate each possible vertex addition.
3) If no vertex additions were performed, generate each possible single-vertex addition.

The condition for single-vertex addition is due to the fact that single-vertex addition does not preserve persistence. Directed vertex addition does. Therefore, these are preferred. Edge insertions are necessary to complete the graph after single-vertex additions are performed. After this process, each of the generated graph operations is executed on the graph $\vec{G}_0$. This process is repeated until all vertices and edges have been added to the graph. Algorithm 2 describes this process. In Algorithm 2, we represent concatenating element s to the end of sequence S by $S \cdot s$.

Figure 4 shows a resulting sequence of this algorithm. We have the following theorem for the effectiveness of this method:

Theorem 10. *For a minimally persistent graph $\vec{G}_n$ with a leader-follower pair, the persistent-Δ generation algorithm will generate a sequence of graph operations that construct $\vec{G}_n$ from a leader-follower seed.*

Algorithm 2 $Persistent\Delta Generation(\vec{G}_n)$

Require: Graph $\vec{G}_n = (V, \vec{E})$ exists such that $\vec{G}_n$ is minimally persistent.
  Initialize leader-follower seed$\vec{G}_0 = (V_0, \vec{E}_0)$ such that $V_0 = \{l, f\}$ and $\vec{E}_0 = \{(f, l)\}$;
  Initialize sequence of graph operations $S = \emptyset$;
  while $card(V_0) < card(V)$ or $card(\vec{E}_0) < card(\vec{E})$ **do**
    Initialize sequence of graph operations $S_i = \emptyset$;
    for all distinct $i, j \in V_0$ **do**
      {Generate all possible edge insertions}
      if $(i, j) \in \vec{E}$ and $(i, j) \notin \vec{E}_0$ **then**
        $ei = edgeInsertion(i, j)$;
        $S_i = S_i \cdot ei$;
      end if
    end for
    $vertexAdded =$ **false**;
    for all $i \in V$ such that $i \notin V_0$ **do**
      {Generate all possible vertex additions}
      if $\exists j, k \in V_0$ such that $(i, j), (i, k) \in \vec{E}$ **then**
        $va = vertexAddition(i, j, k)$
        $S_i = S_i \cdot va$;
        $vertexAdded =$ **true**;
      end if
    end for
    if $vertexAdded =$ **false then**
      for all $i \in V$ such that $i \notin V_0$ **do**
        {Generate all possible single-vertex additions}
        if $\exists j \in V_0$ such that $(i, j) \in \vec{E}$ **then**
          $sva = singleVertex(i, j)$;
          $S_i = S_i \cdot sva$;
        end if
      end for
    end if
    for all operations $op \in S_i$ **do**
      {Perform all determined graph operations}
      for all vertices $i \in op$ **do**
        Add i to V_0;;
      end for
      for all edges $(i, j) \in op$ **do**
        Add (i, j) to $\vec{E}_0$;
      end for
    end for
    $S = S \cdot S_i$;
  end while
  return S;

Proof: Assume that $\vec{G}_n = (V, \vec{E})$ exists, with l, f as the leader and follower of a leader-follower pair, and that $\vec{G}_0 = (V_0, \vec{E}_0)$ is the initialized leader-follower seed. If the graph has only two vertices, the graph is constructed.

If there are more than two vertices, then, by Lemma 3, there exists a path from all vertices in $V \setminus \{l\}$ to vertex l. This implies that there exists a pair of vertices i, j such that $i \in V, i \notin V_0$, $j \in V_0$, $(i, j) \in \vec{E}$. This implies that a single-vertex addition is possible (there may also be vertex additions possible, but this is unnecessary for the proof).

Assume that a single-vertex operation is performed, increasing the size of $V_0, \vec{E}_0 \in \vec{G}_0$. Note that $\vec{G}_0$ always has the leader-follower pair. Therefore, if there are remaining vertices $i \in V$ such that $i \notin V_0$, then Lemma 3 also shows that more single-vertex additions are possible. In fact, more single-vertex additions will always be possible until there does not exist a $i \in V$ such that $i \notin V_0$.

Since we have not added any vertices $i \notin V$ to V_0, this implies that, at this point, $V = V_0$.

For all edges $(i, j) \in \vec{E}$, either $(i, j) = (l, f)$,the leader-follower edge, or (i, j) is not the leader-follower edge. If (i, j) is the leader-follower edge, then it was added to $\vec{E}_0$ when the leader-follower seed was initialized. If it is not the leader-follower edge, note that we have already proven that all vertices V are added to V_0 such that $V = V_0$. This implies that, for any remaining edges not added by vertex or single-vertex additions, there exists a pair of vertices $(i, j) \in V_0$ such that $(i, j) \in \vec{E}$ and $(i, j) \notin \vec{E}_0$. These edges can be added by edge insertions.

Since the algorithm uses these conditions to search for single-vertex additions and edge-insertions, all such operations will be performed, guaranteeing that $V = V_0$ and $\vec{E} = \vec{E}_0$. ■

VII. Conclusions

We have presented a method for determining if target formations for multi-agent networks are persistently feasible, given their maximum sensing and communication range. We introduced an algorithm for generating minimally persistent graphs under proximity constraints. We have also presented new graph operations to construct a persistent graph that represents a formation under range constraints, as well as a method for automatically generating a sequence of these operations for any formation in question. These graphs and operations describe the control and coordination strategies necessary to allow the desired formation to emerge in a multi-agent network.

VIII. Acknowledgements

This work was partially supported under a contract with the National Aeronautics and Space Administration. We also thank Julien Hendrickx for his assistance.

References

[1] G. A. Kaminka and R. Glick, "Towards robust multi-robot formations," in *Conference on International Robotics and Automation*, 2006, pp. 582–8.

[2] K. D. Do and J. Pan, "Nonlinear formation control of unicycle-type mobile robots," *Robotics and Autonomous Systems*, vol. 55, no. 3, pp. 191–204, March 2007.

[3] L. Vig and J. A. Adams, "Multi-robot coalition formation," *IEEE Transactions on Robotics*, vol. 22, no. 4, pp. 637–49, August 2006.

[4] S. Kalantar and U. R. Zimmer, "Distributed shape control of homogeneous swarms of autonomous underwater vehicles," *Autonomous Robots*, vol. 22, no. 1, pp. 37–53, January 2007.

[5] T. Zhijun and U. Ozguner, "On non-escape search for a moving target by multiple mobile sensor agents," in *American Control Conference*, American Automatic Control Council. Minneapolis, MN, USA: IEEE, June 2006, p. 6 pp.

[6] L. Yuan, C. Weidong, and X. Yugeng, "Energy-efficient aggregation control for mobile sensor networks," in *International Conference on Intelligent Computing*, vol. 344. Kunming, China: Intelligent Control and Automation, August 2006, pp. 188–93.

[7] T. Eren, W. Whiteley, B. D. O. Anderson, A. S. Morse, and P. N. Belhumeur, "Information structures to secure control of rigid formations with leader-follower architecture," in *Proceedings of the American Control Conference*, Portland, Oregon, June 2005, pp. 2966–2971.

[8] R. Olfati-Saber and R. Murray, "Graph rigidity and distributed formation stabilization of multi-vehicle systems," *41st IEEE Conference on Decision and Control. Proceedings*, vol. 3, pp. 2965 – 71, 2002.

[9] J. L. A. Jadbabaie and A. S. Morse, "Coordination of groups of mobile autonomous agents using nearest neighbor rules," *IEEE Transactions on Automatic Control*, vol. 48, no. 6, pp. 988–1001, June 2003.

[10] J. M. Hendrickx, B. D. O. Anderson, J.-C. Delvenne, and V. D. Blondel, "Directed graphs for the analysis of rigidity and persistence in autonomous agent systems," *International Journal of Robust and Nonlinear Control*, 2000.

[11] J. M. Hendrickx, B. Fidan, C. Yu, B. D. O. Anderson, and V. D. Blondel, "Elementary operations for the reorganization of minimally persistent formations," in *Proceedings of the Mathematical Theory of Networks and Systems (MTNS) Conference*, no. 17, Kyoto, Japan, July 2006, pp. 859–873.

[12] H. S. M. Coxeter and S. L. Greitzer, *Geometry Revisited*. Washington, DC: Math. Assoc. Amer, 1967, p. 56.

[13] H. Gluck, "Almost all simply connected closed surfaces are rigid," in *Geometric topology, Lecture Notes in Math*, vol. 438. Berlin: Springer, 1975, pp. 225–239.

[14] B. Roth, "Rigid and flexible frameworks," *The American Mathematical Monthly*, vol. 88, no. 1, pp. 6–21, 1981.

[15] T. Tay and W. Whiteley, "Generating isostatic frameworks," *Structural Topology*, no. 11, pp. 21–69, 1985.

[16] G. Laman, "On graphs and rigidity of plane skeletal structures," *Journal of Engineering Mathematics*, vol. 4, no. 4, pp. 331–340, October 1970.

[17] D. J. Jacobs and B. Hendrickson, "An algorithm for two-dimensional rigidity percolation: The pebble game," *Journal of Computational Physics*, vol. 137, no. 2, pp. 346–365, June 1997.