AUTOMATIC COORDINATION AND DEPLOYMENT OF MULTI-ROBOT SYSTEMS

A Dissertation
Presented to
The Academic Faculty

By

Brian Stephen Smith

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy
in
Electrical and Computer Engineering

School of Electrical and Computer Engineering
Georgia Institute of Technology
May 2009

AUTOMATIC COORDINATION AND DEPLOYMENT OF MULTI-ROBOT SYSTEMS

Approved by:

Dr. David Taylor, Committee Chair
Professor, School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Magnus Egerstedt, Advisor
Associate Professor, School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Ayanna Howard, Advisor
Associate Professor, School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Jeff Shamma
Professor, School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Ian Akyildiz
Professor, School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Frank Dellaert
Associate Professor, College of Computing
Georgia Institute of Technology

Date Approved: March 31, 2009

This is dedicated to my parents, Stephen and Renée; to my wife, Mary; and to my daughter, Tori.

ACKNOWLEDGMENTS

First, I want to gratefully acknowledge my advisors, Dr. Magnus Egerstedt and Dr. Ayanna Howard. We would not have been able to accomplish any of our work without their patient instruction and guidance for the last several years. I had a great amount to learn when I arrived at their doorsteps, and they were very generous in response. I especially thank Dr. Egerstedt for showing me the practicality of good theory.

I would also like to thank the committee members: Dr. David Taylor, Dr. Jeff Shamma, Dr. Ian Akyildiz, and Dr. Frank Dellaert. I know that time is a precious commodity, and I appreciate your generosity in this regard.

Gratitude also goes to Dr. John-Michael McNew, Lonnie Parker, Julien Hendrickx, and Jiuguang Wang for their collaboration and inspiration.

On a personal note, I owe an unpayable debt to my parents, Stephen and Renée. I also want to thank my daughter, Tori, for putting up with an often busy father. Last, but not least, I thank my wife, Mary, for being very patient and supportive while I completed this work. I husband could owe no less to another.

This work was supported by the National Aeronautics and Space Administration (NASA).

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

We present automatic tools for configuring and deploying multi-robot networks of decentralized, mobile robots. These methods are tailored to the decentralized nature of the multi-robot network and the limited information available to each robot. We present methods for determining if user-defined network tasks are feasible or infeasible for the network, considering the limited range of its sensors. To this end, we define rigid and persistent feasibility and present necessary and sufficient conditions (along with corresponding algorithms) for determining the feasibility of arbitrary, user-defined deployments. Control laws for moving multi-robot networks in acyclic, persistent formations are defined. We also present novel Embedded Graph Grammar Systems (EGGs) for coordinating and deploying the network. These methods exploit graph representations of the network, as well as graph-based rules that dictate how robots coordinate their control. Automatic systems are defined that allow the robots to assemble arbitrary, user-defined formations without any reliance on localization. Further, this system is augmented to deploy these formations at the user-defined, global location in the environment, despite limited localization of the network. The culmination of this research is an intuitive software program with a Graphical User Interface (GUI) and a satellite image map which allows users to enter the desired locations of sensors. The automatic tools presented here automatically configure an actual multi-robot network to deploy and execute user-defined network tasks.

CHAPTER 1
INTRODUCTION

This work originated with a National Aeronautics and Space Administration (NASA) project to implement a wireless sensor network as a multi-robot system. The premise is that NASA scientists will use a sensor network composed of mobile robots (called *SnoMotes*) to take meteorological sensor readings across ice shelves in Antarctica. These sensor readings will be used as data points to validate global climate models and to better understand the impacts of global climate change.

Working in Antarctica is extremely hazardous and expensive, which makes the use of robots a viable alternative to the use of humans. The NASA scientists should not need to know how to use mobile robots to perform coordinated navigation. Furthermore, manually configuring each robot for each network task becomes expensive, both in terms of time and personnel. Automatic methods for configuring and deploying the SnoMotes are required.

According to specifications, the sensor network should be able to automatically configure and deploy itself to take sensor readings with user-defined spatial resolution, and to report back relevant measurements using communication channels. Also, the robots should be lightweight from a sensing and communications point-of-view. The network should be a decentralized system that requires neither global communication nor global information. Rather, each robot will use its local information in such a manner that allows the entire network to achieve desired network tasks.

In this work, we the present automatic tools for deploying multi-robot networks. Since we desire for our methods to be applicable for a variety of networks (and since the Antarctic sensor platform itself has not been finalized), we consider high-level models of the multi-robot network and its sensor, communication, and locomotion abilities. We define the control systems for the mobile sensor network such that the NASA scientists can use the network with little knowledge of robotic control. Given a multi-robot network capable of sensing, communication, and mobility; its sensing, communication, and mobility parameters (sensor and communication range, maximum velocity, etc.); and a desired network task, the tools automatically configure the network to achieve the network task. This includes tools that allow multi-robot networks to automatically assign individual robots unique roles for the network tasks and to perform them at specific locations in the environment. The culmination of this research is an intuitive software program with a Graphical User Interface (GUI) and a satellite image map which allows users to enter the desired locations of sensors. The automatic tools presented here configure a prototype multi-robot network to deploy and execute these tasks.

In Chapter 2, we introduce the problem in more detail. We also present previous related work in multi-robot systems and multi-agent network control, and present our model of the multi-robot network. We show how many tasks for sensor networks can be described as *formation problems*. We define formations and deployments of multi-robot networks. We also discuss graph-based control, where the topology of the control laws for the network are represented with *network graphs*. Further, we discuss control systems based on *graph grammars* for representing multi-robot networks and their control laws.

Chapter 3 presents preliminary work towards implementing decentralized, graph grammar-based systems for assembling triangulations of robots, as appearing in [1, 2, 3]. Using a preliminary network of *SpiderMote* robots, an automatic control system is defined for achieving triangulations with the robots. This includes definitions for their decentralized control laws, as well as how they switch from one control law to another.

For our multi-robot network, each robot has only limited knowledge of the environment. In fact, each robot can only estimate the relative position of other robots in a local area around it. Therefore, given a desired geometry for the network to achieve, we must determine whether or not the desired geometry is achievable (i.e., *feasible*) for the network. In Chapter 4, we present methods for determining if specific formations are feasible for the multi-robot network, given the limits on its sensing abilities [4, 5]. These methods take into consideration the fact that robots can only estimate the relative position of other robots within a specific *proximity range*. The definition of this proximity range and the desired formation geometry determine if the formation is feasible. Further, these methods automatically generate *persistent network graphs*. Persistent network graphs have many desirable properties which are presented and discussed. These graphs can also be assembled using *graph operations*. We present graph operations for assembling these formations that respect the limited perception of the robots.

Control laws for formations of multi-robot networks with persistent network graphs are presented in Chapter 5 [6]. These control laws assume a network graph generated using the methods in Chapter 4. Using these control laws, robots can estimate their "correct" position in the network using only the local formation geometry and the relative position of two other robots in the network.

While the control laws in Chapter 5 allow the network to *maintain* a formation with little initial error, they do not provide a method for assembling formations. Therefore, an automatic system for assembling formations is defined in Chapter 6 [7, 8]. The system uses the network graphs and graph operations from Chapter 4 and a graph grammar-based system similar to the one in Chapter 3 for assembling feasible formations for the network.

The methods in Chapter 6 assemble a desired formation at the initial location of the network. However, for many network tasks, we desire the network to navigate to a location of interest in the environment before the formation is assembled. This task is complicated if only specific network members have localization ability (i.e., the ability to estimate their location in the environment). In Chapter 7, we show how, despite the lack of localization across the network, robots with localization ability can lead robots without localization ability to satisfy a desired network task [9].

The demonstration in Chapter 7 presents an implementation of all of the results from the previous chapters of this work. Starting with a desired formation and a definition of the network’s sensing/communication proximity range, the methods from Chapter 4 are used to determine that the formation is feasible and to generate a network graph for implementing the formation. The methods in Chapter 7 are used to navigate the robots to the initial location for the formation. Then, the multi-robot network implements a formation assembly as defined in Chapter 6. Once the formation is assembled, the network navigates while maintaining formation using the control laws from Chapter 5. The conclusion of this chapter includes experimental data from the prototype multi-robot network used to verify these methods.

Finally, Chapter 8 concludes this work.

CHAPTER 2

BACKGROUND

In this chapter, we discuss the problem of coordinating and deploying multi-robot systems. We also discuss previous work concerning multi-robot systems in formations.

Many network tasks that the NASA scientists will request require the multi-robot system to be "spaced" at certain intervals to achieve a desired sensor resolution. For example, measuring the energy across a field of ice at 1 m resolution requires each sensor to be positioned within 1 m of its nearest "neighboring" sensors. Similarly, defining specific locations for multiple sensors in the environment implicity defines the relative geometry of the sensors to each other. Therefore, many network tasks can be seen as *formation problems*, in which the goal for the multi-robot system is to achieve specified geometric relationships between the robots at a location of interest. Formations of multi-robot systems have received much recent attention (see [10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25] for a representative sample).

2.1 The Multi-Robot System

For the multi-robot network, we define $n \in \mathbb{N} : n \geq 2$ as the number of robots in the multi-robot network. This defines an index set $N = \{1, \ldots, n\}$ for the network. Thus, robot i is the robot indexed by $i \in N$. We consider the network over an interval of time defined by $T = [0, \infty)$.

Networks of mobile robots are complicated systems, and the dynamics of mobile robots varies from one robot to another. We are concerned with the complexity of multi-robot networks at a high-level. Therefore, we use a kinematic model to represent the multi-robot network. Since these are planar robots, the positions of the robots are modeled by states in two dimensions such that, $\forall i \in N$, $x_i : T \mapsto \mathbb{R}^2$ represents the *trajectory* of robot i, and, $\forall t \in T$, $x_i(t)$ is location of robot i at time t. We represent the control for each robot as a

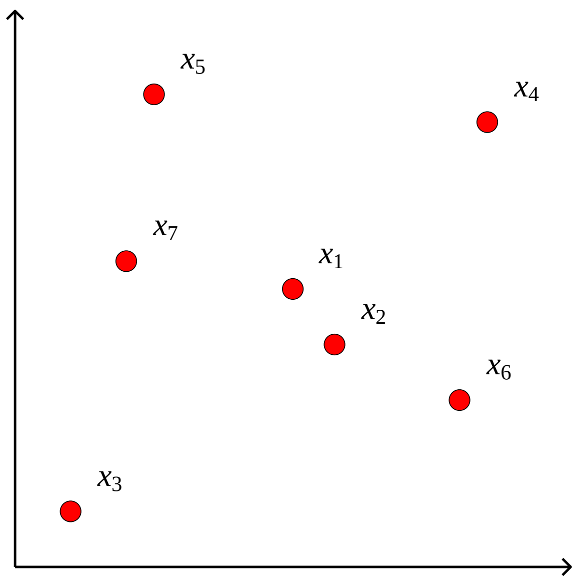


Figure 2.1. Multi-robot network . Each robot is represented as a point in $\mathbb{R}^2$.

single integrator such that, $\forall i \in N$, $u_i : T \mapsto \mathbb{R}^2$ is the control for robot i and, $\forall i \in N$, $\dot{x}_i(t) = u_i(t)$.

The entire network is represented by a *network trajectory* $X : T \mapsto \mathbb{R}^{2n}$ such that, $\forall t \in T$, $X(t) = \left[x_1(t)^T, \ldots, x_n(t)^T\right]^T$. For all pairs of robots $(i, j) \in N \times N$, the distance between robots i and j at time $t \in T$ is represented by $\left\|x_i(t) - x_j(t)\right\|$, where $\|\ldots\|$ denotes the Euclidian norm. Figure 2.1 depicts an example multi-robot network.

2.2 Desired Formations and Deployments

Qualitatively, we say that a solution to a formation task for a network of mobile robots has three goals:

1. Assembly: To control the network to achieve a state such that each robot satisfies desired, relative geometric relationships with the other network members.

2. Maintenance: Once the network has assembled the formation, to maintain the formation. This includes maintaining the formation as the network moves from one place to another.

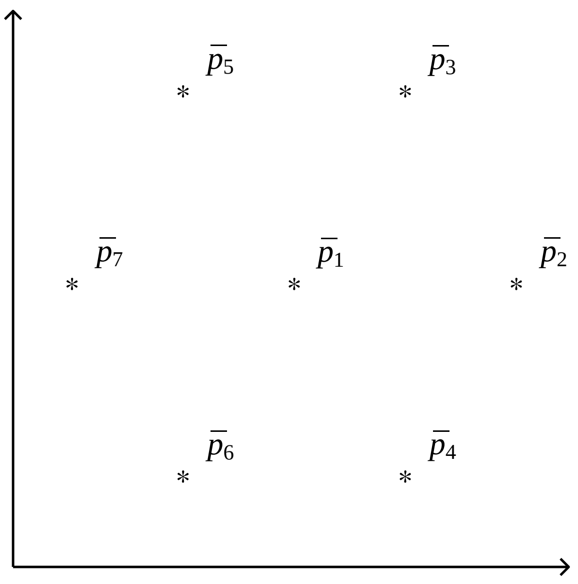


Figure 2.2. Formation positions. The "shape" of a *desired formation* is defined by a set of *positions* in $\mathbb{R}^2$. This is an example formation for the multi-robot network in Figure 2.1.

3. Deployment: To control the network such that the robots satisfies given, relative geometric relationships with each other at a desired location of interest. This may be a static location in the environment, or a location that changes with time.

To define a desired formation, we define *n formation positions* indexed by N such that position i is the position indexed by $i \in N$. The positions of the formation are also modeled as points in two dimensions such that, $\forall i \in N$, $\bar{p_i} \in \mathbb{R}^2$ is position i. The entire *desired formation* is represented by $\bar{P} \in \mathbb{R}^{2n}$ such that $P = \left[\bar{p}_1^T, \ldots, \bar{p}_n^T\right]^T$. Figure 2.2 depicts the shape of an example desired formation.

The geometric relationship between the formation positions defines the "shape" of the formation. Note that position i neither defines a desired location for robot i, nor is it required that robot i is assigned to position i, for this would reduce the problem to stabilizing each robot to its corresponding desired position. Rather, accomplishing formation assembly involves assigning each robot a position in the formation and controlling the robots such that the shape of the network matches the shape of the desired formation. In other words, there must exist an assignment function $a : N \mapsto N$ that maps each robot to a unique position. Further, the robots' control laws must be defined such that, $\forall (i, j) \in N \times N$,

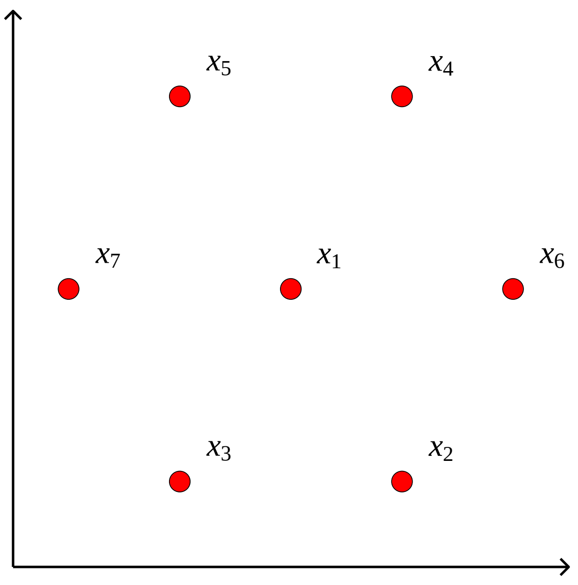


Figure 2.3. Formation assembly. The goal of formation assembly for the multi-robot network in Figure 2.1 with the desired formation in Figure 2.2.

$\left\|x_i(t) - x_j(t)\right\| \to \left\|\bar{p}_{a(i)} - \bar{p}_{a(j)}\right\|$ as $t \to \infty$. Figure 2.3 depicts the assembly of the desired formation depicted in Figure 2.2 by the multi-robot network depicted in Figure 2.1.

A *desired deployment* describes the location in the environment where the network members should be placed. To this end, we model *n deployment positions* indexed by indices N such that, $\forall i \in N$, $p_i : T \mapsto \mathbb{R}^2$ models the desired location of a network member. In other words, $p_i(t)$ is the location where a unique network member should be at time $t \in T$. We assume that, $\forall i \in N$, that p_i is continuously differentiable. Further, we assume that, $\forall (i, j, t) \in N \times N \times T$, that $\|p_i(t) - p_j(t)\|$ is constant (i.e.,$\forall (i, j, t) \in N \times N \times T$, $\|p_i(t) - p_j(t)\| = \|p_i(0) - p_j(0)\|$). These deployment positions define the network's *deployment trajectory* $P : T \mapsto \mathbb{R}^{2n}$ such that $P(t) = \left[p_1(t)^T, \ldots, p_n(t)^T\right]^T$. The deployment trajectory describes both a desired formation shape ($\bar{P} = P(0)$) and the desired location of the formation in the environment, which may be a function that varies with time. Figure 2.4 shows an example trajectory in which the network traverses an area of interest while in formation. Figure 2.5 shows the network performing the desired network task.

Note that a network deployment can be defined in a variety of ways. For example, the NASA scientists themselves can explicitly specify the locations that the network should

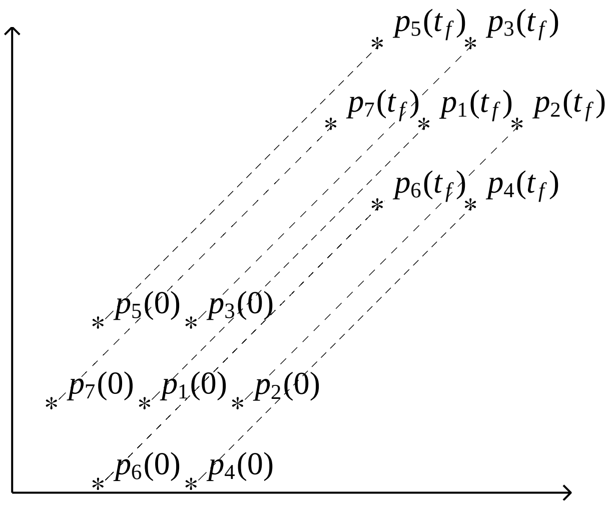


Figure 2.4. Deployment example. We assume that $0 < t_f$. These positions define the desired network trajectory as a function of time. To satisfy a formation, we insist that, $\forall(i, j, t) \in N \times N \times T$, $\|p_i(t) - p_j(t)\| = \|p_i(0) - p_j(0)\|$.

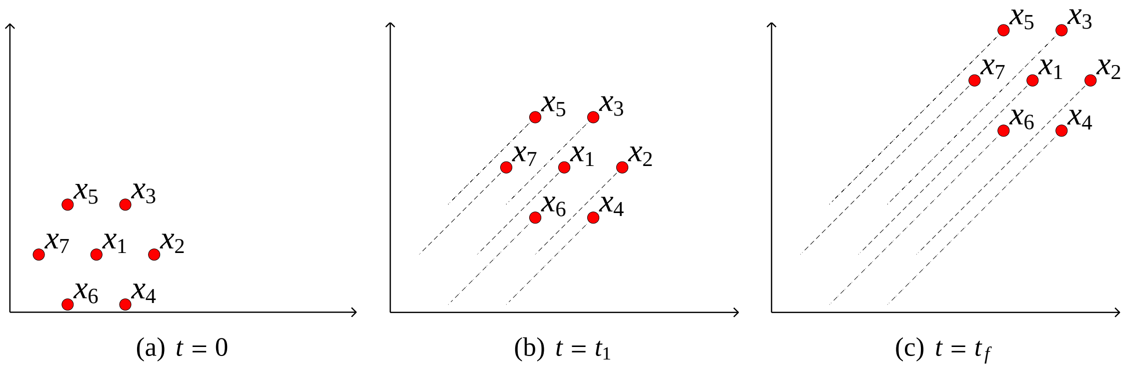


Figure 2.5. Deployment implementation example. This figure shows a network accomplishing the network task defined by the deployment in Figure 2.4. In Figures 2.5(b) and 2.5(c), the dotted lines indicate the previous positions of the robots.

satisfy as a function of time. Alternatively, the scientists could specify the network to "track" a specific, moving feature in the environment with a specific relative formation. In this example, the motion of the feature defines the network deployment.

2.3 Sensing and Communication Limitations

As discussed in Chapter 1, the robots in the network have limitations on their sensor and communication abilities. Here, we present the sensor and communication limitations of the network.

2.3.1 Localization Limitations

We describe *localization ability* as the ability of a robot to estimate its position in the environment. For example, a robot with a GPS sensor, an inertial navigation system, and rotary encoders on its wheels can estimate both its location and heading relative to the earth, as well as to a reference location and heading.

If all network members have localization ability, then the problem can be solved by assigning each robot a unique deployment position and stabilizing each robot to its corresponding deployment position. However, such an approach has drawbacks. First, since the method hinges on localization, such a method is not implementable with a multi-robot network in which any network members do not have localization. This also implies that the loss of localization of a robot results in the failure of that robot; it cannot participate in any formations that require it to be at a different location. Secondly, synchronizing a multi-robot network such that the robots follow a specific trajectory simultaneously is difficult with large networks. Finally, it is unclear how effectively the inter-robot geometry is preserved in the case of sensor noise and errors, since the robots would ignore their relative geometry during the execution.

As an alternative, we consider many real-world examples of formations in systems which have no localization ability (for example, flocks of birds, formations of airplanes, etc.). In these systems, formations are deployed using only the *relative positions* of the network members. This implies that multi-robot networks can assemble formations even if localization ability is denied to some or all of the network.

For the multi-robot network we consider here, we assume that only a proper subset of network members have localization ability. This can be due to a light-weight design, for we show that many network tasks can be accomplished with little or no localization. This can also arise out of network failures (e.g., the failure of the localization sensors of a subset of robots). We do not want such limited sensor failure for a proper subset of robots to imply complete network failure (the inability of the multi-robot network to accomplish any network tasks). Therefore, the methods in this work consider limited or no localization for the multi-robot network.

2.3.2 Proximity Sensor and Communication Limitations

For our multi-robot network, we assume that there are limits in terms of which robots can sense and communicate with each other. We also assume that this corresponds to the relative geometry of the robots in the network. Therefore, the sensor and communication limitations are modeled by a *proximity function* $\mathbb{F}(i)$, defined such that $\mathbb{F}(i) \subseteq N$ is the set of indices of robots that robot i can sense and communicate with. Note that the definition of $\mathbb{F}$ is determined by the sensing parameters of the multi-robot network in question, and can vary from network to network.

The sensors that robots use to estimate the relative positions of nearby robots are complex. Often, robots use a variety of sensors simultaneously, each with its own parameters. Again, since we are concerned with the complexity of multi-robot networks at a high-level, we use a simplified proximity function. For the network we consider here, a *proximity range* $\Delta > 0$ is defined such that $j \in \mathbb{F}(i)$ at time $t \in T$ if and only if $||x_i(t) - x_j(t)|| \leq \Delta$.

2.4 Decentralized Control

Due to the limited sensor and communication abilities of the robots, the sensor network is implemented as a *decentralized system*, as stated in Chapter 1. For multi-agent systems, *centralized control strategies* are strategies in which a centralized control agent plans for and controls all robots in the system. It has been shown that planned, centralized control strategies for multiple-robot systems are computationally difficult, even when assuming that the robots themselves are the only dynamic feature in the environment [26]. Also, centralized control strategies typically require global network information to implement. This information may be unavailable or difficult to obtain, especially considering the sensor limitations of the network we consider here. Therefore, much attention is given to *decentralized* control strategies, in which each robot makes decisions based on local information in a manner that allows global goals to be achieved. Decentralized strategies have also received much attention (e.g., [15, 22, 26, 27, 28, 29]). Such control strategies are often described as employing *nearest neighbor rules* to achieve their global objectives. These nearest neighbor rules define the control laws for each agent based on their nearest neighbors.

Decentralized strategies are inspired partially from biological systems, such as flocks of birds and schools of fish, which achieve advantageous global behaviors without the presence of any centralized intelligence. In [27], methods are presented for simulating the behavior of flocks of "boids," where the control for each "boid" is determined by the spatial relationship between each of its "nearby flockmates." It also combines several flocking behaviors (collision avoidance, velocity matching, and flock centering) to produce the individual control for each "boid". When implemented, a global behavior emerges that greatly resembles a flock of birds in flight.

Some of the earliest results in multi-robot formation control are included in [10], in which robots are modeled as spheres in three-dimensional space, and a set of points in three-dimensional space represents a formation pattern. In [10], several strategies for formation movement are proposed, including nearest-neighbor tracking and multi-neighbor tracking, as well as leader-based formations. Work such as [10] and [27] laid the groundwork for [30], in which several formations used by US Army scouts are implemented with a team of four robots. The contribution of [30] is the combination of formation and navigation behaviors that allow robot teams to navigate to a specific location while avoiding hazards and maintaining formation. However, these methods do not consider the sensor and communication limitations we consider in this work. Also, the methods in [30] implement specific example formations used in military applications. As such, it does not address how to automatically handle arbitrary, user-defined formations for the network that we consider.

2.4.1 Graph-Based Control

As previously mentioned, decentralized control strategies implemented with multi-agent networks frequently feature robots using nearest neighbor rules, in which each agent has specific relationships with a proper subset of other agents. Therefore, it is often convenient to use *graphs* to represent these networks. Specifically, graphs are used to represent the topology of the control laws, as well as the information available to each agent. Graph-based modeling of decentralized control strategies has received much recent attention (for example, see [13, 14, 17, 18, 28, 29, 31, 32, 33, 34]).

In this work, the network control law topology is modeled with a *network graph* $G(t) = (V, E(t))$. Here, V is the *vertex set*, and $E(t)$ is the *edge set*. We use a notation that, for a graph $G(t) = (V, E(t))$, $V(G(t)) = V$, and $E(G(t)) = E(t)$. For a network with n robots, $|V(G(t))| = n$. As such, $V(G(t))$ is indexed by indices N such that $V(G) = \{v_1, \ldots, v_n\}$. Each vertex is associated by index to its corresponding robot such that, $\forall i \in N$, $v_i \in V(G(t))$ is the vertex of robot i. Each edge $e \in E(G(t))$ is an ordered pair such that $e = (v_i, v_j) \in V(G(t)) \times V(G(t))$. As such, $G(t)$ is a *directed graph*. In this example, v_i is the *tail* of edge $e \in E(G)$, and v_j is the *head* of e. The existence of $(v_i, v_j) \in E(G(t))$ indicates a control law relationship between robots i and j (i.e., the control laws for robot i are a function of the position of robot j). Also, $E(t)$ is a function of time, as robots may break and/or establish control law relationships as the system evolves. For notational clarity, we “drop” the explicit time dependence of $G(t)$ when it is not relevant, describing the network graph as simply G.

If there is a directed path from $v_i \in V(G)$ to $v_j \in V(G)$ in G, we say that robot i is a *predecessor* of robot i, and robot j is a *successor* of robot i. Similarly, if there exists edge $(v_i, v_j) \in E(G)$, we say that robot i is an *immediate predecessor* of robot j, and robot j is an *immediate successor* of robot i.

Each robot is modeled as a hybrid automaton that is in a specific *mode* at a given time $t \in T$. Therefore, the topological model of the network is augmented to a *vertex-labeled graph*. In these graphs, a label set Σ is defined corresponding to all the possible modes of the robots, and $l : V(G) \times T \mapsto \Sigma$ is a *labeling function* that indicates the mode of each robot at time $t \in T$. In this manner, a labeled network graph is a quadruple: $G(t) = (V, E(t), \Sigma, l(t))$. The double $(G(t), X(t))$ models both the network topology, the modes of the robots, and network geometry at time $t \in T$. Figure 2.6 shows an example network graph. Each edge $(v_i, v_j) \in E(G(t))$ is depicted as an arrow between states x_i and x_j. The vertices are also labeled by mode, where each mode corresponds to the geometry the robot must satisfy with its adjacent neighbors.

In this work, $\mathbb{R}^+$ defines the set of strictly positive real numbers. The graph $G(t)$ and the network trajectory $X(t)$ define a *length function* $\delta : V(G(t)) \times V(G(t)) \times \mathbb{R}^{2n} \mapsto \mathbb{R}^+$ such that, $\forall (v_i, v_j) \in V(G(t)) \times V(G(t))$, $\delta(v_i, v_j, X(t)) = \|x_i(t) - x_j(t)\|$. We define $\delta_{ij} = \delta(v_i, v_j, X(t))$ for simpler notation. Thus, each edge $(v_i, v_j) \in E(G(t))$ has an assigned weight δ_{ij}, and G is a *weighted graph*.

In order to model the topology of the information available to each robot at a high level, we use an undirected *proximity graph* $\mathbb{G}(t) = (V, \mathbb{E}(t))$. Note that $V(\mathbb{G}(t)) = V(G(t))$, the vertex set in the network graph. Thus, $\forall v_i \in V(\mathbb{G}(t))$, vertex v_i corresponds to robot i. In the proximity graph, an edge (v_i, v_j) exists in $E(\mathbb{G}(t))$ if and only if $j \in \mathbb{F}(i)$ at time t, implying that robot i can sense and communicate with robot j. Figure 2.7 shows a Δ-disk proximity graph, where an edge exists between each pair of vertices $(v_i, v_j) \in V(\mathbb{G}(t)) \times V(\mathbb{G}(t))$ if and only if $\|x_i(t) - x_j(t)\| \leq \Delta$.

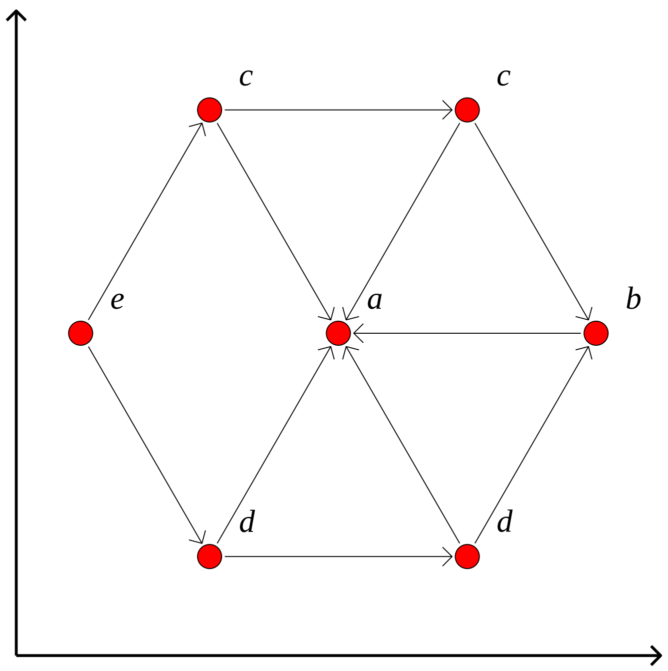


Figure 2.6. An example *network graph* is represented by placing vertices at the locations of the robots and placing edges between the vertices. These edges represent the topology of the control laws. The label set is $\Sigma = \{a, b, c, d, e\}$. In this example, each label corresponds to each robots's mode based on the geometry it must satisfy with its neighbors.

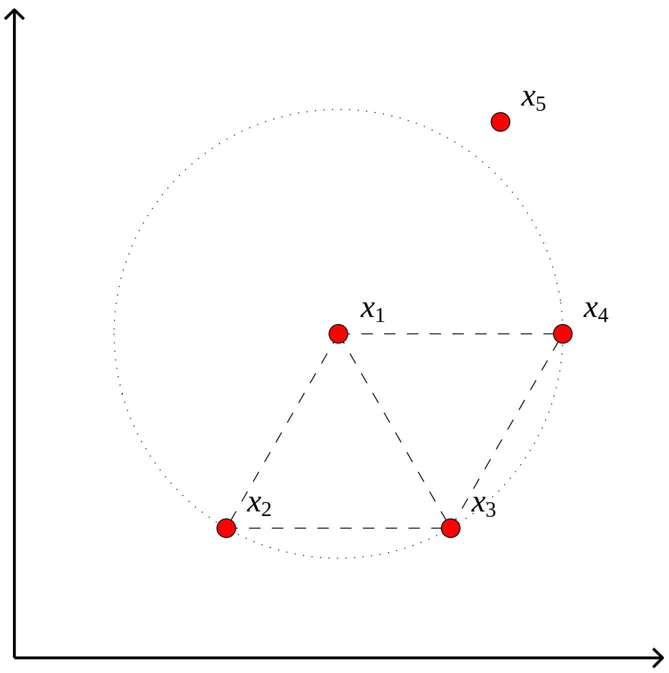


Figure 2.7. Δ-disk proximity graph $\mathbb{G}(t)$. The circle around x_1 represents the area in which robot 1 can sense and communicate with other robots. Dashed lines indicate edges in $E(\mathbb{G}(t))$, corresponding to pairs of robots that can sense and communicate with each other. Robot 5 is too far away to sense or communicate with any other robot.

2.4.2 Consensus Problems

While such a system is fully described by its equations, studying the topology associated with these systems has yielded very useful results. The Laplacian of the adjacency matrix of multi-agent system graphs has been extensively exploited (see [13, 18, 28]). Both [13] and [18] describe how the eigenvalues of the Laplacian of the adjacency matrix can show the stability of systems with nearest neighbor control laws. In [28], the properties of the Laplacian are exploited to model the system presented in [27] and prove the convergence of the velocities of the boids. Both [27] and [28] are examples of *consensus* (or *agreement*) *problems*, in which the goal is to achieve convergence to a common value for each agent in the network. Another such consensus problem is rendezvous, in which the goal is convergence to a common location. The rendezvous problem has been explored heavily (e.g., [29, 31, 34, 35]). It has been shown that formation control problems can be expressed as consensus problems in which the formation error is defined for each robot, and the goal is for each robot's formation error to stabilize to zero [29, 35].

A variety of consensus and formation control problems have been solved if generous assumptions are made concerning the information available to each robot. For example, [28] assumes that the control topology is connected "frequently enough" to allow for successful convergence, and [29] assumes that the control topology is modeled as a connected graph. However, due to sensor and communication limitations, these assumptions are not guaranteed for the multi-robot network that we consider.

In this work, we assume that a robot cannot effectively execute a control law without the information required to implement it. Therefore, another goal for the system is guaranteeing that there exists an edge $e \in E(\mathbb{G}(t))$ if there exists the same edge $e \in E(G(t))$. This is often described as *maintaining* or *preserving connectivity*.

In [13, 18, 31], necessary and sufficient connectivity conditions are established for the stability of multi-agent systems for consensus problems. However, these works do not suggest methods for guaranteeing these connectivity conditions. Other work has been done to develop methods to preserve sufficient connectivity, if it is initially present (see [32, 34, 35]). However, these methods often hinge on broad multi-agent cooperation [32] and/or localization [34, 35]. In [32], it is assumed that all agents coordinate their choice of control values across the network in order to guarantee connectivity, an approach that does not scale well with large networks. In [34, 35], it is assumed that the agents in the multi-agent system have an "agreement" as to their orientation relative to a common reference frame, as well as the desired orientation of the formation in that frame. For our multi-robot network, this implies that the robots can estimate their location and/or heading relative to some common reference frame corresponding to the environment, which is beyond the abilities we assume for the network. Other examples of robots assembling formations have been recently presented, (e.g., [15, 25, 36]), but these methods also require either global communication ability, global network information, network-wide localization, and/or centralization to implement.

2.5 Rigidity, Persistence, and Henneberg Sequences

A source of inspiration towards maintaining and assembling formations of robots under our assumptions of limited sensing and communication is the concept of *persistence*, which is closely related to *rigidity*. Here, we define and discuss *rigid and persistent networks*.

2.5.1 Rigid Networks

In a *rigid network*, a formation of robots is maintained by *directly* maintaining only a subset of inter-robots distances, but in a manner which preserves all inter-robot distances. The concept of rigidity has received much attention (see [19, 37, 38, 39, 40]). Rigidity was first studied as it pertains to *frameworks* (structures of flexible joints) which are also modeled with graphs. In a rigid control strategy, given a network graph G, each edge $(v_i, v_j) \in E(G)$ models a *constraint* for robots i and j, implying that they must maintain their inter-robot distance $\|x_i(t) - x_j(t)\|$. The network trajectory X is *edge-consistent* if and only if the inter-robot distances corresponding to edges in G are preserved, i.e., $\|x_i(t) - x_j(t)\| =$

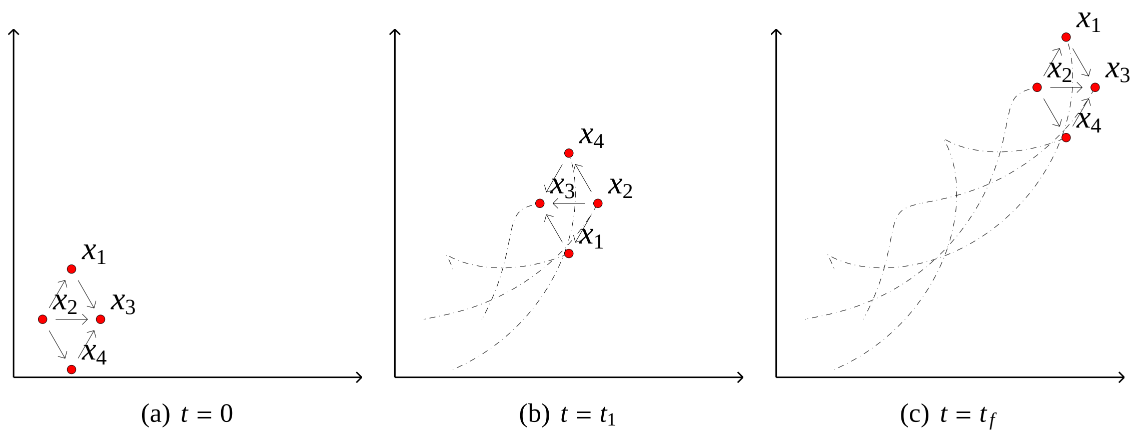


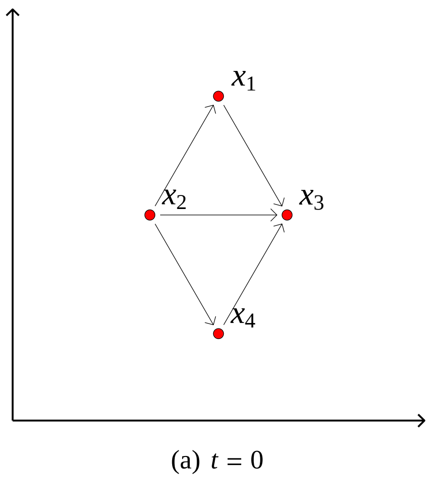
Figure 2.8. A rigid network. We assume that $0 < t_1 < t_f$. Here, robots maintain the inter-robot distances corresponding to the edges in the network graph. This preserves the formation, allowing only translations and rotations of the formation.

$\|x_i(0) - x_j(0)\| \; \forall((v_i, v_j), t) \in E(G(t)) \times T$. The network trajectory X is *rigid* if and only if all inter-robots distances are preserved (whether or not they correspond to edges), i.e., $\|x_i(t) - x_j(t)\| = \|x_i(0) - x_j(0)\| \; \forall(i, j, t) \in N \times N \times T$. The network itself is *rigid* if all edge-consistent trajectories are rigid trajectories. If the network is not rigid, it is *flexible*.

Rigid networks can maintain their formation by maintaining only the inter-robot distances corresponding to the edge set in G. This suggests a useful strategy for maintaining a formation by only *directly* maintaining a proper subset of inter-robot distances. Figure 2.8 shows a trajectory of a rigid network.

2.5.2 Persistent Networks

A rigid network must preserve the constraints in the network, which correspond to the edges in the network graph. As such, rigidity itself does not consider the direction of the edges in the network. In a *persistent network* [40], these directions are important. For each pair of robots whose inter-robot distance is to be directly maintained, the responsibility of maintaining the distance is delegated to a single robot of the pair. Hence, each edge $e \in E(G)$ is an ordered pair such that $e = (v_i, v_j)$, representing a *constraint* of robot i only (not of robot j). In this case, robot i has a constraint with robot j, and robot i attempts to maintain its distance from robot j. Such a constraint does not imply anything for the



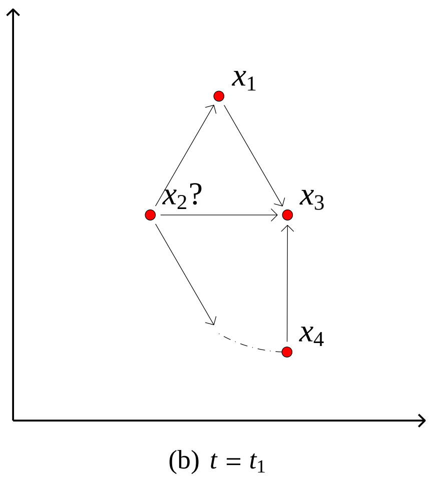
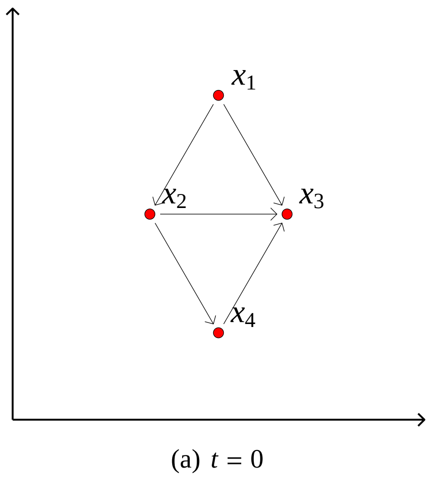


Figure 2.9. An example of constraint inconsistence. We assume that $t_1 > 0$. If robot four performs circular motion around robot three, agent two cannot satisfy all three constraints.



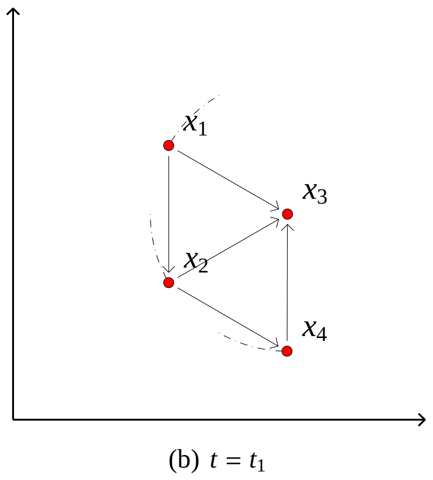


Figure 2.10. An example of constraint consistence. We assume that $t_1 > 0$. If robot two performs circular motion around robot three, robots one and four can still satisfy their constraints

control laws of robot j. In a persistent network, each robot has constraints with all of its immediate successors in the network graph.

Qualitatively, a network is *constraint consistent* if the directions of each edge are oriented such that no robot can satisfy its constraints in a manner which forces another robot to violate a constraint [40]. The constraint consistence of a network is determined by the out-degree of the vertices in its network graph (see [40] for a more thorough treatment). Figure 2.9 shows a constraint inconsistent network, while Figure 2.10 shows a constraint consistent network.

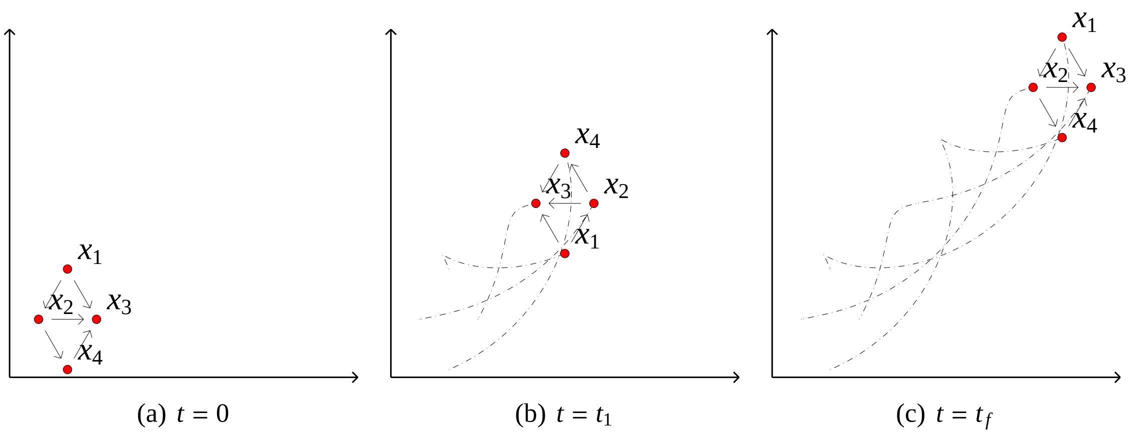


Figure 2.11. A persistent network. We assume that $0 < t_1 < t_f$. The robot at the tail of each edge is responsible for maintaining the corresponding inter-robot distance.

The network in Figure 2.8 is rigid, but not constraint consistent. *A persistent network is both rigid and constraint consistent* [40]. Figure 2.11 shows a persistent network. Rigid and persistent networks are discussed in more detail in Chapter 4.

2.5.3 Previous Results with Persistent Networks

Control laws for persistent networks have been demonstrated in [16, 20]. In these strategies, for each robot i, assume that $J(i)$ is the set of all immediate successors of robot i (i.e., $J(i)$ is the set of indices $j \in N$ such that $(v_i, v_j) \in E(G)$). For a minimally persistent network in two-dimensions, this implies that $|J(i)| \leq 2$ [40]. For a desired formation $\bar{P}$ and $\forall i \in N$, dynamics of robot i are defined in [16, 20] et al. as

$$\dot{x}_i(t) = \sum_{j \in J(i)} -\left(\|x_i(t) - x_j(t)\| - \|\bar{p}_i - \bar{p}_j\|\right)\left(x_i(t) - x_j(t)\right). \tag{1}$$

Stability analysis reveals that these control laws are asymptotically stable at two equilibrium points for each robot except when the locations of all three robots are collinear [16, 20]. However, many formations can only be satisfied if each robot stabilizes to a specific equilibrium point. These dynamics do not guarantee a particular equilibrium point, and it can be shown by example that a network can start in formation but “lose” its formation if sufficient error is introduced. Such an error can occur when the formation is moving from one location to another. This can result in a network that behaves as a flexible network despite its rigidity. Also, these dynamics do not address the issue of assembling or maintaining the formation with the sensor limitations that we consider in this work.

2.6 Embedded Graph Grammar Systems (EGGs)

A source of inspiration towards assembling formations is the *Embedded Graph Grammar (EGG)* [41, 42], a formalism that encodes dynamic, geometric, and network properties of a multi-agent system in a unified manner. At the core of the EGG is the notion of a *graph grammar* that takes as inputs vertex-labeled graphs, and produces other vertex labeled graphs according to a given rule set. Through the application of the rules in the rule set, edges may be removed or added to the graph, and the vertex labels may change.

In an EGG system, a set of *graph transition rules* Φ defines how the network graph can change topology and labeling, which represents how the robots in the network can change modes and their control laws. Each rule $r \in \Phi$ is defined as a pair such that $r = (L \rightharpoonup R)$, where L is a vertex-labeled *left graph* and R is a vertex-labeled *right graph*. When a label-preserving induced subgraph isomorphism exists between the left graph L of a rule and the network graph G, the induced subgraph of G can be replaced by the right graph R of the same rule. This rule application involves adding and/or removing edges and changing labels, corresponding to robots changing their control law relationships and their modes.

Each rule also has an associated *guard function* g, which evaluates to *true* or $false$ depending on if the states associated with the induced subgraph of G satisfy certain (often geometric) constraints. These guard functions are used to model limitations on when a rule can be applied, and typically model the sensing limitations of the robots. For example, in this work, the guards are defined to evaluate to $false$ if the robots' locations are too far apart, indicating that the robots cannot sense and communicate with each other.

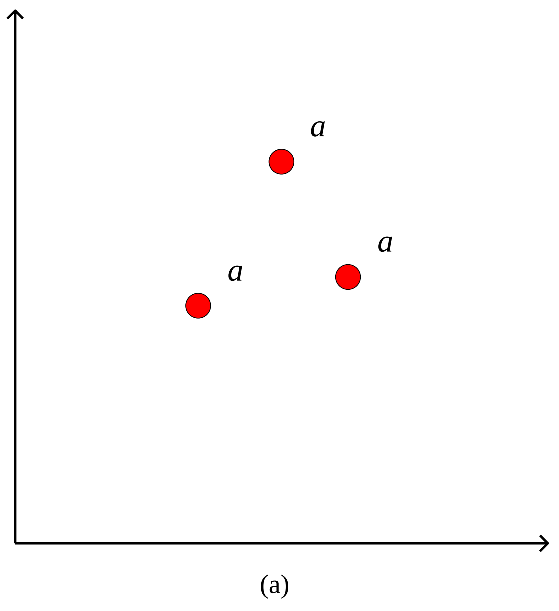
EGG systems are non-deterministic in that, at any given time, multiple rules can be applied simultaneously. EGG systems insist, however, that no robot can be involved in simultaneous rule applications. In other words, the sets of robots in each simultaneous rule application must be disjoint.

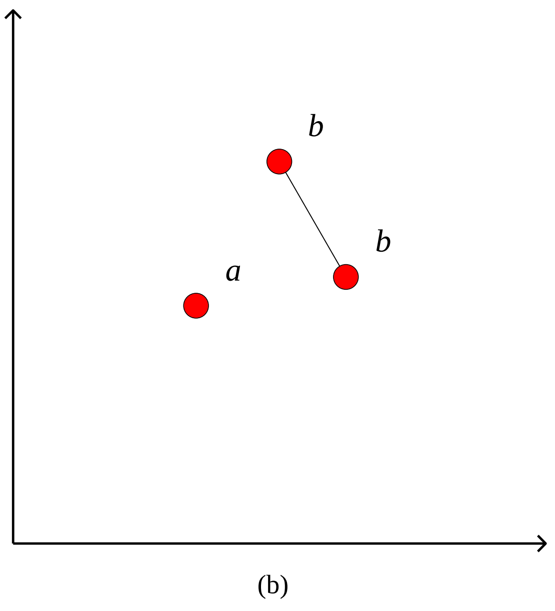
As an example, consider the rule set

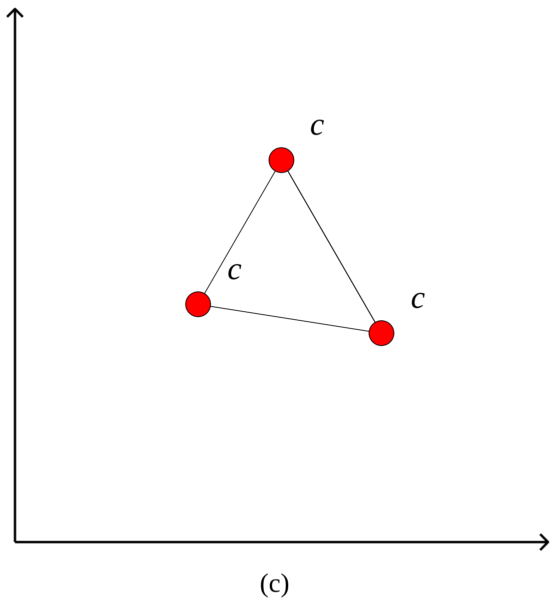
$$\Phi = \begin{cases} a \quad a \rightharpoonup b - b & (r_1), \\ \begin{array}{ccc} & b & \\ b & \diagup & \quad a \end{array} \rightharpoonup \begin{array}{ccc} & c & \\ \diagup & & \diagdown \\ c & \text{——} & c \end{array} & (r_2), \end{cases} \tag{2}$$

with the network depicted in Figure 2.12(a). Assume that the control laws of b and c are defined such that each robot stabilizes to a state that is a distance $d > 0$ away from any adjacent robot in the network graph. Assume that all robot can sense and communicate with each other, and that the guards are always satisfied. Then rule r_1 in (2) can be and is applied, producing the network graph shown in Figure 2.12(b). The robot labeled b attempt to make their distance d, and rule r_2 in (2) is applied, producing the network graph in Figure 2.12(c). The robots attempt to set all inter-agent distances to d, and eventually stabilize to an equilateral triangle formation, as shown in Figure 2.12(d).

EGG systems have been defined that solve a variety of assembly and coverage problems [41, 42]. However, few of these systems have been implemented with real robotic systems. In Chapter 3, we present an implementation of an EGG system on an actual multi-robot network.







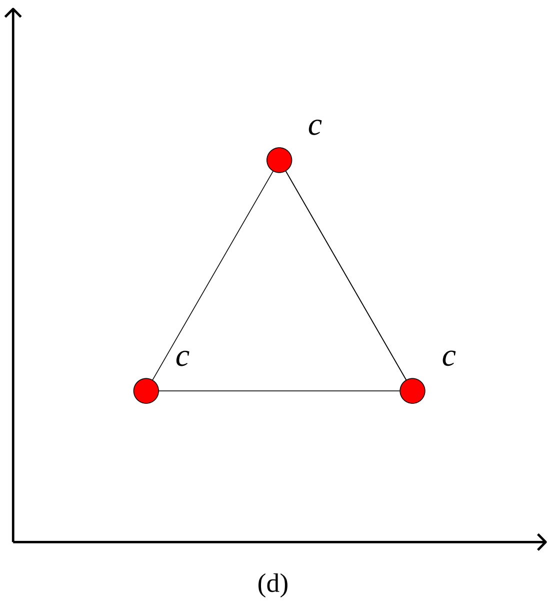


Figure 2.12. An Embedded Graph Grammar (EGG) example to assemble an equilateral triangle formation. In these figures, the *labels* are displayed, showing the mode of each robot. In Figure 2.12(a), each robot starts out with label a. In Figure 2.12(b), two robots apply the first rule in (2), analogous to assembling one side of the triangle. In Figure 2.12(c), the second rule in (2) is applied. Figure 2.12(d) shows the final equilateral triangle formation.

CHAPTER 3

MULTI-ROBOT COORDINATION WITH EMBEDDED GRAPH GRAMMAR SYSTEMS (EGGS)

As mentioned in Chapters 1 and 2, we employ many graph-based methods for automatically configuring and deploying the multi-robot network. In this chapter, we discuss work towards implementing Embedded Graph Grammar (EGG) systems with a multi-robot network. A variety of EGG systems have been proposed, but few have actually been implemented on actual robotic systems. Here, a preliminary multi-robot network is introduced, composed of *SpiderMotes,* a precursor of the SnoMotes. The SpiderMotes are described, as well as methods for implementing EGG systems with them. We present an effective communication protocol that allows EGG systems to be implemented on decentralized systems. We show that the effective design of an EGG system for a network of robots is directly related to the design and abilities of the robots in the network.

3.1 Mobility Platform: The SpiderMotes

As stated in Chapter 1, the robots in the multi-robot network for NASA must be integrated sensor/actuation platforms that combine communication and sensing capability with mobility. Such integrated platforms allow for controlled repositioning of the sensor devices such that measurements at desired spatial and temporal resolution can be achieved. Based on NASA's specifications, the attributes that must be included in the robots' design are the following:

- The translational and rotational velocities of the robots (v and ω) must be controllable.
- Each robot must possess a communication channel such that it has the ability to receive and transmit information to other robots.

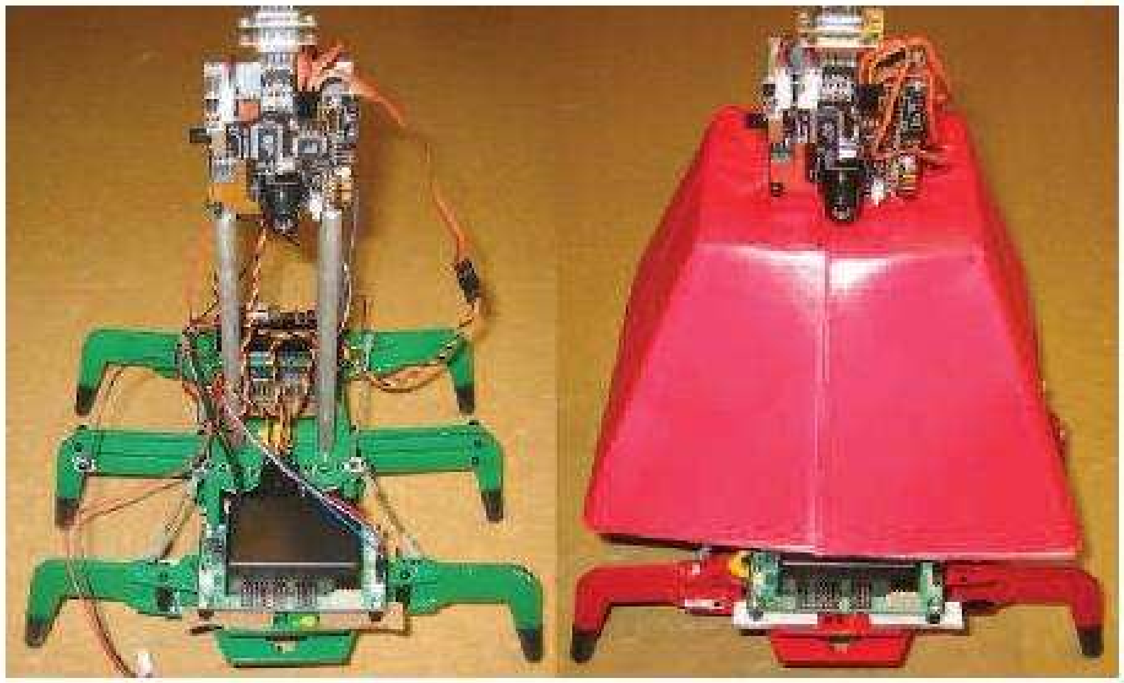


Figure 3.1. A *SpiderMote* robot. The robot on the left shows the chassis with equipped hardware. The robot on the right shows the color-coded cover, allowing other robots to sense and identify this robot.

- Each robot must possess a sensing channel such that it has the ability to verify the presence and location of other sensor nodes within the network.
- Each robot must have on-board computation such that it can function independently and does not require a centralized host for decision-making.

This generic definition of a the network member allows us to define the network without being limited to the utilization of specific robotic hardware, communication, or sensing technology. Figure 3.1 depicts our rendition of a robot in the sensor network that qualifies under this definition: the *SpiderMote*. The SpiderMote is a custom-made platform that consists of a hexapod chassis, CMUCam2 vision sensor, wireless communication module, and controller.

The skeletal body of the platform has dimensions of $[7.5\times26.5\times24]$ cm. The controller and battery pack are mounted on the rear of the chassis, while the wireless communication module is mounted to the front. The CMUCam2 is mounted on a 15 cm boom. Also,

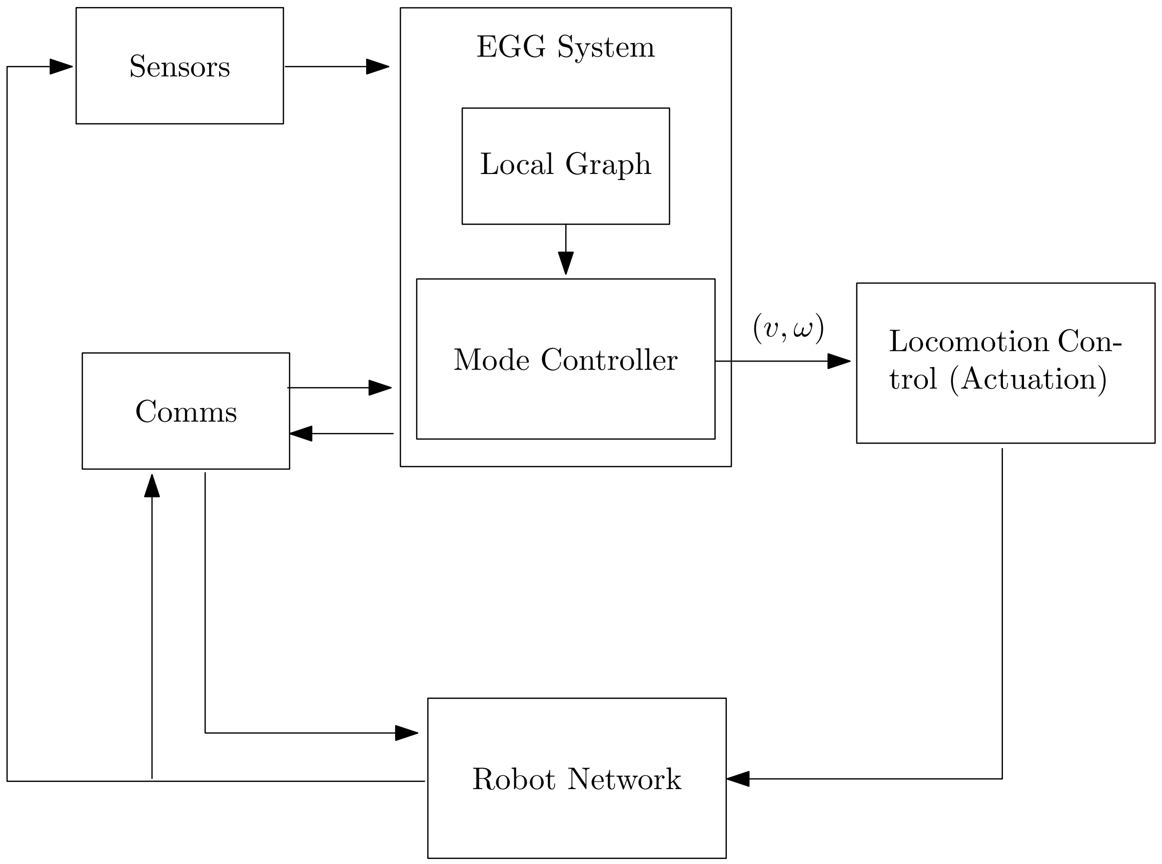


Figure 3.2. Control system overview. For each robot in the network, the EGG system generates the appropriate translation velocity *v* and angular velocity ω for the robot.

each robot is equipped with a color-coded tetrahedral cover, which is mounted below the camera. Figure 3.2 presents a system overview of the SpiderMote.

3.1.1 Communications

The wireless communication module is capable of transmitting and receiving a single 10 byte message at any given time. There are two types of messages: requests and responses. Because messages could potentially be lost, requests are reattempted after a timeout period until the appropriate response is received. This allows the SpiderMotes to exchange information and coordinate their behaviors. These are used to implement the communication protocol presented in Chapter 3.2.

3.1.2 Perception/Sensing

To allow the robots to perceive each other, each SpiderMote recognizes others by color and estimates their positions based on image data. Each robot's chassis is tagged with a specific color, such as red, blue, and green. The relative distance and angle is estimated based on the location of the colored pixels representing each robot. (Note that this choice is made to facilitate an easy algorithm for identifying adjacent robots, but more robust schemes can certainly be envisioned.)

The CMUCam2 has a field-of-view (FOV) of $\approx 0.28\pi$ rad, centered in the direction of the robot's heading. When the camera is set to track a specific color, it returns a tracking data packet in the form of an 8-tuple: $\left(m_x, m_y, x_1, y_1, x_2, y_2, pixelcount, confidence\right)$, which represents information about the tracked pixels, which are pixels in the camera image that match the color being tracked. The pair (m_x, m_y) represent the center-of-mass of the tracked pixels in the image, while the pairs (x_1, y_1) and (x_2, y_2) represent the corners of the smallest bounding box that contains all tracked pixels, as shown in Figure 3.3. The *pixelcount* represents the number of tracked pixels. The $confidence$ is a ratio between the pixel count and the number of pixels in the bounding box. These values are used to estimate the position of robots in the field-of-view with an accuracy of $\approx$ 10 cm.

The SpiderMotes serve as a platform for implementing the networked control algorithms needed to solve the sensor coverage problem, as well as others. Next, we discuss how EGGs are implemented on the SpiderMotes.

3.2 Implemented Embedded Graph Grammars

As presented in Chapter 2.6, Embedded Graph Grammars (EGGs) are mathematical models that produce non-deterministic trajectories. They do not assume any particular communication architectures. When an embedded graph grammar is implemented on a real system, a low-level protocol that implements the grammar must be designed.

Since the label and adjacency information is distributed across the network such that

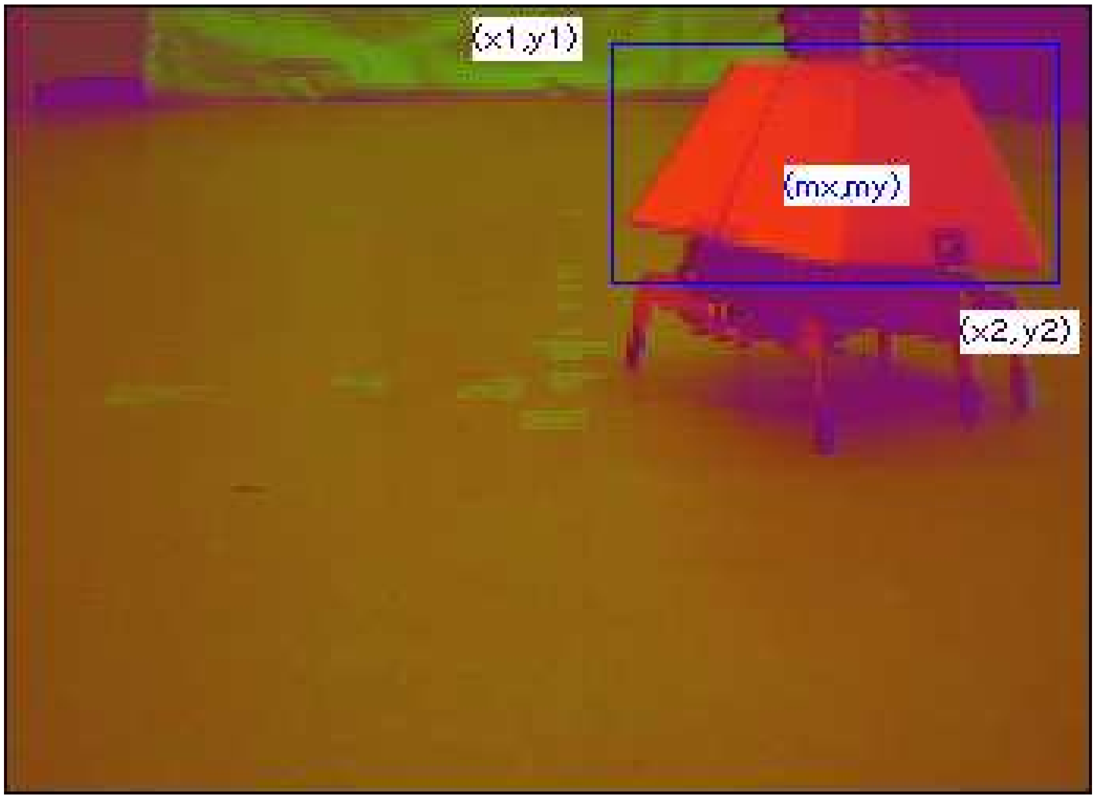


Figure 3.3. Example camera view, showing center-of-mass and bounding box coordinate locations.

each robot has immediate access only to its own label and adjacency information, the label and adjacency information corresponding to other robots can only be obtained through communication. For an EGG to be successfully executed, the network must change labels and adjacency information in a manner defined by the EGG. Since this is a decentralized network, this implies that robots must negotiate rule applications in a manner consistent with the EGG. It is necessary to ensure that each robot applying a rule has exclusive control of the local network graph information of each robot involved in the rule application; if not, then it is possible for robots to modify the graph information in a manner inconsistent with the rules in the rule set.

Before we move on to actually defining the protocol, we note that in many cooperative control methods, topology switches are deterministic, often determined by geometry, e.g. [33]. Often in these systems, the robots may switch their control modes independently. Embedded graph grammars do not fit into this category due to their inherent non-determinism.

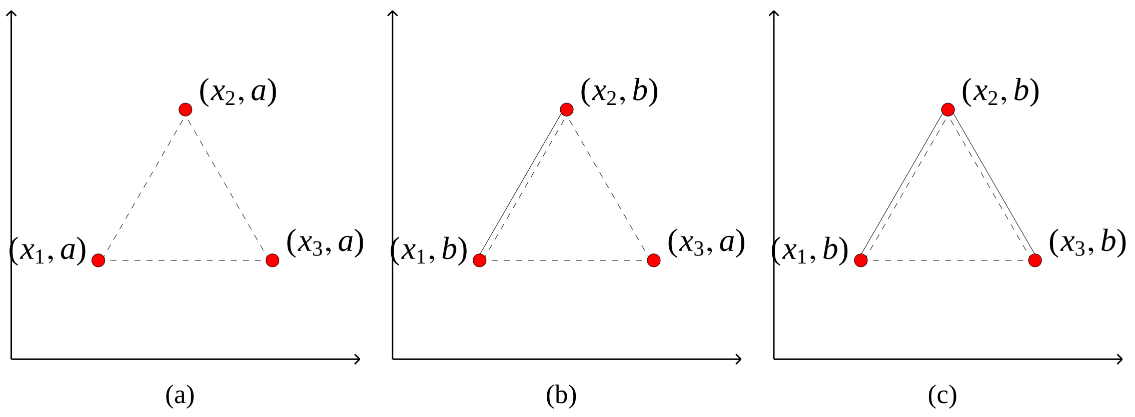


Figure 3.4. Graph inconsistency example. Figure 3.4(a): The rule can be applied by robot pairs $\{(1,2),(1,3)\} \in N \times N$. Figure 3.4(b): The rule is applied with robot pair(1,2). Figure 3.4(c): The rule applies also with (1,3), producing a network graph not intended by the rule.

This is easy to see by considering the graph in Figure 4(a), and the corresponding EGG system from Chapter 2.6:

$$\Phi = \begin{cases} a \quad a \quad \rightharpoonup \quad b - b & (r_1), \\ \begin{array}{ccc} & b & \\ b & \diagup & a \end{array} \quad \rightharpoonup \quad \begin{array}{ccc} & c & \\ \diagup & & \diagdown \\ c & \text{——} & c \end{array} & (r_2). \end{cases} \tag{3}$$

In Figure 3.4, $\forall i \in N$, robot i is labeled as a double (x_i, label) where *label* corresponds to the label of vertex v_i in the network graph G. In Figure 3.4, the solid lines correspond to the edges in the network graph G, dashed lines correspond to edges in the proximity graph $\mathbb{G}$. Suppose that without agreeing, robots 1 and 3 apply the rule $r_1 \in \Phi$ from (3) with robot 2 as shown in Figure 3.4. The resulting graph transformation shown in Figure 3.4(c) is inconsistent with the valid trajectories of the grammar. We call this a *graph inconsistency*.

A protocol that allows groups of robots to come to *agreement* before applying a rule is required. We briefly contrast peer-to-peer protocols (*explicit agreement*) with simple token protocols (*implied agreement*). We then present an *extended token* protocol for the network.

3.2.1 Single Token and Peer-to-Peer Protocols

The question of *deadlock* arises in any system requiring agreement. Deadlocks occur when the communication protocol does not allow any rules to be applied even though some rules

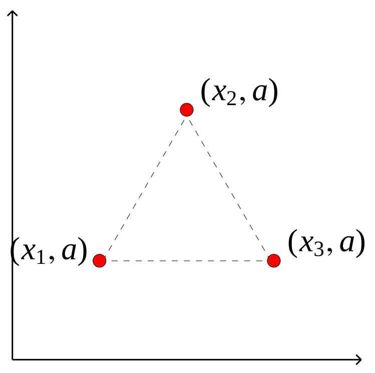


Figure 3.5. Deadlock example. If all the robots try to apply the rule $r_1 \in \Phi$ shown in (3), the network can deadlock.

are applicable. Consider a protocol where each robot independently attempts to apply the rule $r_1 \in \Phi$ defined in (3), and then communicates a request for rule application to the other robot in the rule application. Further, assume that a robot may then either accept or deny the request, and will always deny a request if it is in the process of sending one. Suppose for the graph in Figure 3.5, robot 1 identifies robot 2 for rule application, robot 2 identifies robot 3 for rule application, and robot 3 identifies robot 1. Each robot will send a request to the counterclockwise robot and deny a request from its clockwise robot, resulting in deadlock despite the fact that multiple applications are possible.

The question of deadlock has a rich literature. One simple method to avoid deadlock is a single token architecture where the robot possessing the token is the only one capable of deciding which rule to apply and updating the network. These architectures often initialize with some form of leader election algorithm.

Contrast this with a rule application protocol appropriate for peer-to-peer communication. In such a protocol, random communication requests establish temporary links between pairs of robots. The pairs of robots pass messages to elect a leader, who requests label information from the other robot in the pair and then decided if the rule is applicable. Since multiple pairs may identify and apply rules independently, the result is highly parallel rule processing. However, the search is not complete since any cut of the graph does not examine all possible pairings. Thus, no rule may be applied even though some rules are applicable. The peer-to-peer protocol reduces the space complexity for large systems since a vertex only needs to maintain its local topological information and not a copy of the entire graph. However, the communication complexity is increased since messages must be passed to elect the leader and share the local topology.

We capture some of the parallel processing abilities of the peer-to-peer architecture as well as guarantee a *deadlock-free* evaluations of the rules in an architecture called *extended token-based rule protocol*. This protocol takes advantage of the "broadcast" capabilities of the system to distribute processing among the robots resulting in a *nearly concurrent* system.

3.2.2 Extended Token-Based Rule Protocol

Rather than having the robot with the token single-handedly evaluate the rule set and apply a rule before then passing the turn to another robot, we instead construct a type of token that distributes the rule evaluation. We call this *extended token-based rule protocol*. Figure 3.6 shows an example iteration of this protocol.

An *extended token* consists of a random sequence of network indices. This extended token determines the order in which robots are allowed to apply rules. We can express an extended token of a network with n robots as (n) where (n) is a sequence of the network indices N. For example, if $n = 5$, a token could be $(5) = (5, 4, 2, 1, 3)$.

Also, the network has a *used set*, which the robots use to keep track of what robots have been used in recent rule applications. Each robot keeps track of the used set by network broadcasts. By updating the used set and avoiding the application of rules involving robots in the used set, we guarantee that graph inconsistencies are avoided.

We describe this protocol as an *iterative protocol,* in which a new extended token is generated for each iteration. Initially, we designate a single robot to generate the first extended token and broadcast it to the network. We also initialize the used set of each robot to empty. This begins the first iteration. During each iteration, the following takes place:

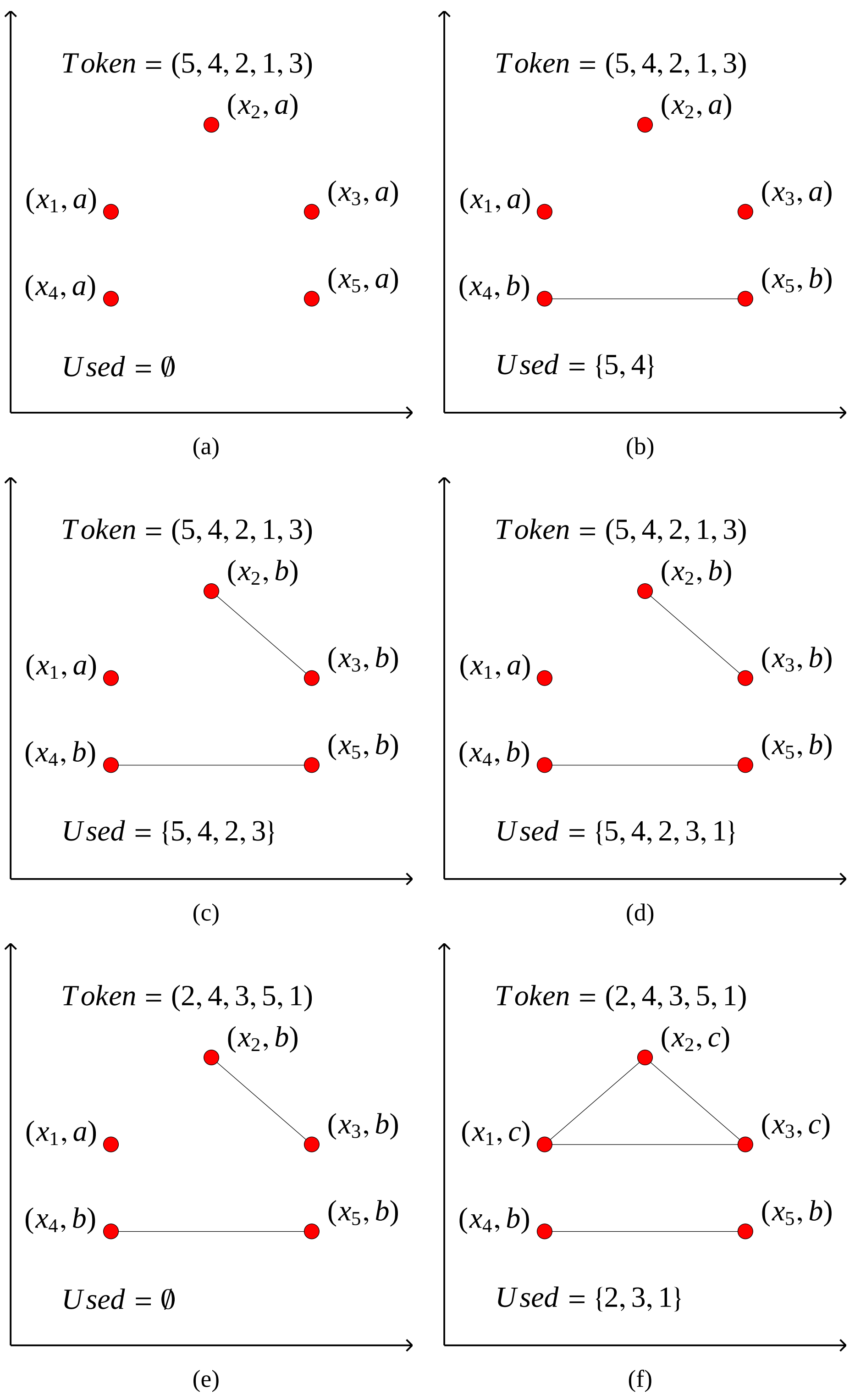


Figure 3.6. An Extended token-based rule protocol example iteration. We define the rule set as in (3). The rule $r_1 \in \Phi$ is initially applicable for every pair of robots. Figure 3.6(a): Initial setup. The used list is empty, it is robot 5's turn to apply a rule. Figure 3.6(b): Robot 5 applies r_1 with robot 4. The network graph edges are updated, as well as the used set. Figure 3.6(c): Robot 2 can now apply a rule. It applies r_1 with robot 3. Figure 3.6(d): Robot 1 cannot apply any rules, it broadcasts itself as used. Figure 6(e): The token iteration is complete. Robot 3 now generates a new token and broadcasts it to the network. This clears the used set, and another iteration begins. Figure 3.6(f): Robot 2 applies rule $r_2 \in \Phi$. No more rules can be applied.

1. All robots clear their used set such that it is empty.

2. All robots consider all previous network graph information out-of-date.

3. Each robot concurrently searches the partition of the rule search space that includes itself and the robots that follow it on the token, requesting updated information from those robots.

4. The robot with the lowest index in the extended token which is also not in the used set attempts to apply any applicable rules involving itself as follows:

   - If a rule is applicable and none of the involved robots are in the used list, it applies the rule, changing the adjacencies and/or labels in the network graph. It then broadcasts the robots involved in the rule application to the network, who add them to the used set. Since it can only apply a rule that applies to itself, it broadcasts itself as used.

   - If no rules are applicable, it broadcasts itself as used, and each robot in the network adds it to the used list.

5. The last robot in the extended token generates a new random extended token, and broadcasts it to the network. This begins a new iteration.

During each iteration, the robot whose turn it is to apply rules is always the robot with the lowest index in the extended token who has also not been used during the current iteration. Thus, robots take their turn as the used list updates are broadcast. Since this guarantees that there is never more than one robot actively attempting to a apply rule, the vertex sets of their rule applications are always disjoint (i.e., no robots are ever involved in multiple, simultaneous rule applications in the same iteration). This also guarantees that deadlock is avoided, since the robots take turns having exclusive control of the network graph information. In other words, if we assume that the vertex sets of the rule applications are not disjoint, or that deadlock is occurring, this would violate our definition of the protocol.

Since all graph information is “thrown away” by each robot at the beginning of each iteration with the cleared used set, and since the used set is avoided for all rule applications during each iteration, this ensures that graph inconsistencies do not occur, since this guarantees that the robots involved in any rule application have not changed their labels or adjacencies since the rule was evaluated as applicable. In other words, if we assume that a graph inconsistency takes place, this implies that a robot in a rule application changed its graph information after its graph information was obtained. This implies that it is in the used set, which violates our definition of the protocol.

Finally, the search of the rules is “complete” only when no rules are applicable. For every token, if *any* rule is applicable, at least one rule will be applied. The resulting implementation exhibits nearly concurrent behavior.

3.3 Experimental Results

To implement an EGG system on a networked system of $n = 3$ SpiderMotes, we first describe how we model the SpiderMotes, their control laws, and the implemented EGG system. Then, we present the results of executing the EGG system by the SpiderMotes.

3.3.1 Dynamics and Control Laws

The robots are indexed by index set $N = \{1, 2, 3\}$. Each robot in the network can be described by the unicycle model such that, $\forall i \in N$, $(x_i, y_i)^T \in \mathbb{R}^2$ and $\phi_i \in [-\pi, \pi)$ are the position and orientation of robot i. Also, $\forall i \in N$, $v_i \in \mathbb{R}^+$ and $\omega_i \in \mathbb{R}$ are the controlled translational and rotational velocities of robot i such that

$$\begin{aligned} \dot{x}_i &= v_i \cos(\phi_i), \\ \dot{y}_i &= v_i \sin(\phi_i), \\ \dot{\phi}_i &= \omega_i. \end{aligned}$$

For all $(i, j) \in N \times N$, we define the angle of robot j to robot i as $\rho_{ij} = tan^{-1}\left(\frac{y_j - y_i}{x_j - x_i}\right)$. Sensor information from the camera provides the robots with the relative displacement between robots within its angular field of view of $\pm\vartheta = \pi/7$ rad centered around $\phi_i\ \forall i \in N$. Therefore, the information available as control support for each robot is given by the pair

$$\begin{aligned}\delta_{ij} &= \sqrt{(x_j - x_i)^2 + (y_j - y_i)^2}\\ \theta_{ij} &= \rho_{ij} - \phi_i,\end{aligned}$$

where $\delta_{ij} \in \mathbb{R}^+$ represents the distance between robots j and i, and $\theta_{ij} \in [-\pi, \pi)$ represents the relative angle of robot j to the heading ϕ_i of robot i. This implies that, $\forall (i, j) \in N \times N$, robot j is perceivable by robot i if and only if $|\theta_{ij}| \leq \vartheta$. To accurately represent the network, the control laws of each robot must be a function only of the relative position of robots it can perceive.

In order to decrease the distance between all robots, thereby increasing the possibility of robots being able to perceive and communicate with each other, a control law for bringing robots closer together was developed for robots with limited perception. For all $i \in N$, we define the proximity function $\mathbb{F}$ such that $\mathbb{F}(i) = \{j \in N : |\theta_{ij}| \leq \vartheta\}$. In other words, $\mathbb{F}(i)$ is the set of indices corresponding to robots in the FOV of robot i. The positions of these robots define a centroid with a distance $\delta_i^{\mathbb{F}(i)}$ and relative angle $\theta_i^{\mathbb{F}(i)}$ to the position and orientation of robot i such that

$$\begin{aligned}\delta_i^{\mathbb{F}(i)} &= \sqrt{\left(\frac{\sum\limits_{j\in\mathbb{F}(i)} \delta_{ij}\cos(\theta_{ij})}{|\mathbb{F}(i)|}\right)^2 + \left(\frac{\sum\limits_{j\in\mathbb{F}(i)} \delta_{ij}\sin(\theta_{ij})}{|\mathbb{F}(i)|}\right)^2}\\ \theta_i^{\mathbb{F}(i)} &= \tan^{-1}\left(\frac{\sum\limits_{j\in\mathbb{F}(i)} \delta_{ij}\sin(\theta_{ij})}{\sum\limits_{j\in\mathbb{F}(i)} \delta_{ij}\cos(\theta_{ij})}\right).\end{aligned}$$

We define upper bounds on the allowable error in position and orientation as ϵ_δ and ϵ_θ, respectively. There are derived experimentally, based on the accuracy of the robots in moving and estimating the relative locations of other robots. For all $i \in N$, we define the

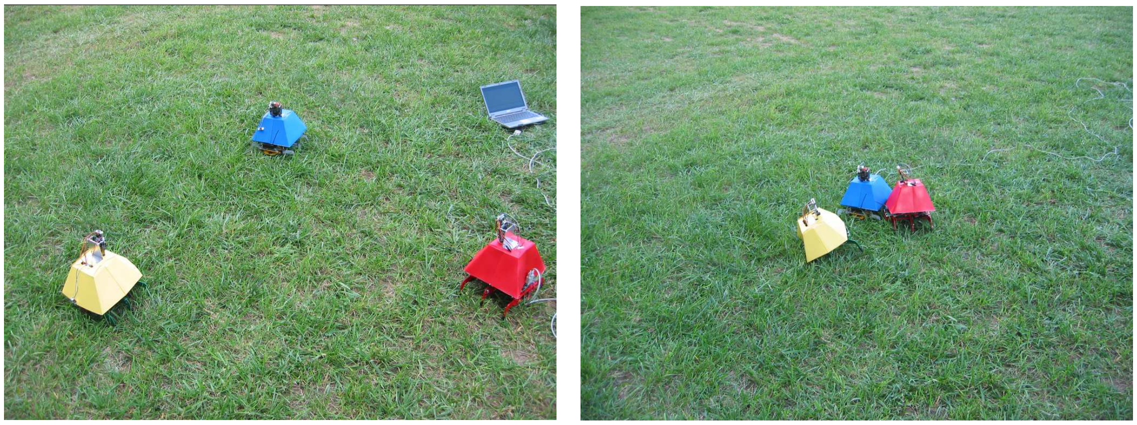

(a) $goTo$: initial setup (b) $goTo$: complete

Figure 3.7. $goTo$ **executed outdoors**

control law $(v_i, \omega_i) = goTo(i)$ such that

$$v_i = \begin{cases} v_0 \cdot \delta_i^{\mathbb{F}(i)} & \text{if } \mathbb{F}(i) \neq \emptyset \text{ and } \delta_i^{\mathbb{F}(i)} \geq \epsilon_\delta, \\ 0 & \text{otherwise} \end{cases}$$

$$\omega_i = \begin{cases} \theta_i^{\mathbb{F}(i)} & \text{if } \mathbb{F}(i) \neq \emptyset \text{ and } \delta_i^{\mathbb{F}(i)} \geq \epsilon_\delta \text{ and } |\theta_i^{\mathbb{F}(i)}| \geq \epsilon_\theta, \\ 0 & \text{if } \mathbb{F}(i) \neq \emptyset \text{ and } \delta_i^{\mathbb{F}(i)} \geq \epsilon_\delta \text{ and } |\theta_i^{\mathbb{F}(i)}| < \epsilon_\theta, \\ \omega_0 & \text{otherwise} \end{cases},$$

where i is the vertex of the robot performing $goTo$ and v_0 and ω_0 are positive constants. If no robot is perceived by robot i, robot i stops and rotates until a robot is perceivable. Figure 3.7 shows robots performing $goTo$. Figure 3.7(a) shows three robots at their initial setup. During all experiments such that three robots execute this control law within perception range of each other, the distances between each robot decrease until they are directly next to each other, as shown in Figure 3.7(b).

To allow a robot to achieve a desired distance to a specific robot, the control mode $(v_i, \omega_i) = setDistance(i, j, d)$ is defined as

$$v_i = \begin{cases} v_0(\delta_{ij} - d) & \text{if } j \in \mathbb{F}(i) \text{ and } |\delta_{ij} - d| \geq \epsilon_\delta, \\ 0 & \text{otherwise} \end{cases}$$

$$\omega_i = \begin{cases} \theta_{ij} & \text{if } j \in \mathbb{F}(i) \text{ and } |\theta_{ij}| \geq \epsilon_\theta, \\ 0 & \text{if } j \in \mathbb{F}(i) \text{ and } |\theta_{ij}| < \epsilon_\theta, \\ \omega_0 & \text{if } j \notin \mathbb{F}(i) \text{ (i.e., otherwise)} \end{cases},$$

where i is the vertex of the robot performing *setDistance*; v_0, ω_0, ϵ_δ, and ϵ_θ are the previously defined constants; d is the desired distance; and j is the index of the robot that robot i is trying to achieve a distance d to. If robot j is not perceivable, then robot i stops and rotates until robot j is perceivable.

3.3.2 EGG System Definition

The EGG used for the assembling triangular formations is the quadruple $(G_0, \Phi, \mathbb{F}, u)$.

1. G_0: The initial network graph $G(0) = G_0$ is defined such that there are no edges and each vertex has a label a.

2. Φ: The rule set is defined as in our example:

$$\Phi = \begin{cases} a \quad a \quad \rightharpoonup \quad b - b, & (r_1) \\ \begin{tikzcd}[ampersand replacement=\&] \& b \arrow[dl, dash] \& \& \& \& c \arrow[dl, dash] \arrow[dr, dash] \& \\ b \& \& a \& \rightharpoonup \& c \arrow[rr, dash] \& \& c \end{tikzcd} & (r_2). \end{cases} \qquad (4)$$

   The control laws for these labels are defined in Table 1.

3. $\mathbb{F}$: In this network, the members have global broadcast ability. Therefore, the proximity function captures the limited visual perception of the robots. The proximity function $\mathbb{F}$ spawns a proximity graph $\mathbb{G}$ such that a directed edge (v_i, v_j) exists in $E(\mathbb{G})$ if and only if robot j is in the FOV of robot i, i.e. $j \in \mathbb{F}(i)$. Since the network has global communication ability, the proximity graph $\mathbb{G}$ has an undirected edge between each pair of vertices in $\mathbb{G}$. The guard for rule $r_1 \in \Phi$ requires that a directed

Table 1. Control Laws for Mode by Label

LABEL l(i)	MODE
a	$(v_i, w_i) = goTo(i)$
b	$(v_i, w_i) = setDistance(i, j, d_b)$
c	Hybrid control law in Figure 3.8

and undirected edge exists between the corresponding vertices in $\mathbb{G}$ for the rule to be applicable. The guard for rule $r_2 \in \Phi$ requires that all robots in the rule application be fully connected with undirected edges. Further, the robot labeled a in the left graph be able to perceive one of the robots labeled b in the network graph. This implies that there is a directed edge in $\mathbb{G}$ between the vertices corresponding to the robot in mode a and one of the robots in mode b for the rule to be applicable.

4. u: The locally implementable mode controller u for each robot i is a function of the label associated with the corresponding vertex i in G. The control modes by label are defined in Table 1. The control law for mode a is $goTo(i)$. For mode b, $setDistance(i, j, d_b)$ is executed such that j is the index of the robot paired with the robot i as in the right side of rule r_1 in (4), and the constant d_b represents the desired distance to maintain between robots adjacent robots labeled b. This brings two robots in mode b to the distance d_b from each other. The control law for mode c is a hybrid one shown in Figure 3.8 that alternately performs *setDistance* on robots j and k adjacent to the robot in question in the right side of the rule $r_2 \in \Phi$ from (4). Once the distance d_c is achieved for one robot, it switches to the other, and then repeats.

3.3.3 Execution

Here, we describe two executions of the defined EGG. The first execution uses two robots, while the second execution uses three.

3.3.3.1 "Bar" formation in a two robot system

Two robots as described previously are configured to execute the defined EGG. We define the length of the "bar" to be $d_b = 1.5$ m to allow the execution to be easily photographed.

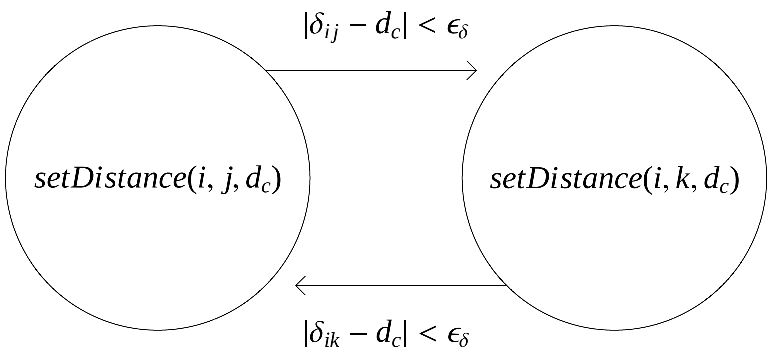

$|\delta_{ij} - d_c| < \epsilon_\delta$
$setDistance(i, j, d_c)$
$setDistance(i, k, d_c)$
$|\delta_{ik} - d_c| < \epsilon_\delta$

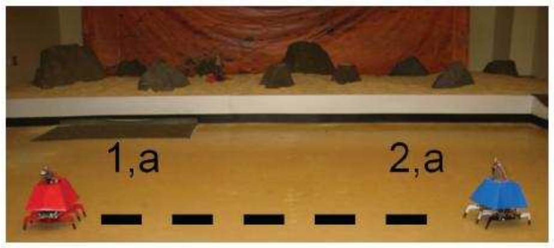
Figure 3.8. Hybrid control law for mode *c*.

Each robot begins with the label a and begins performing $goTo$ in control mode a. Since they begin without being able to sense another robot, they begin searching for a robot by rotating and processing sensor data from the camera, as shown in Figure 3.9(a). Since each robot can communicate with each other, they begin the token-based rule execution scheme, transmitting tokens, as well as label, adjacency, and perception information. Since no robot can sense the other,there are no sensing edges in the proximity graph. The guards prohibit any rule for being applied.

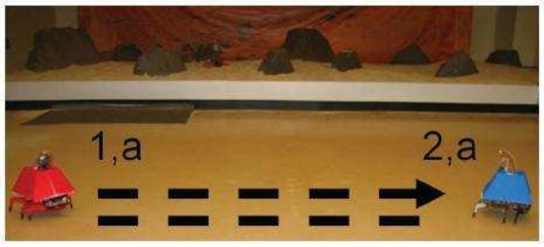
Eventually, one robot is able to sense the other, creating a sensing edge in the proximity graph, as seen in Figure 3.9(b). Since now the guard is satisfied and the robots match the left side of $r_1 \in \Phi$ in (4), the rule can be applied. The robot whose turn it is currently in the token-based rule protocol communicates the rule application to the other. This creates an edge in the network graph G, indicated by the solid line in Figure 3.9(c). The robots begin executing the label b control laws, and eventually settle at a distance d_b from each other, as indicated in the same figure.

3.3.3.2 Equilateral triangle formation in a three robot system

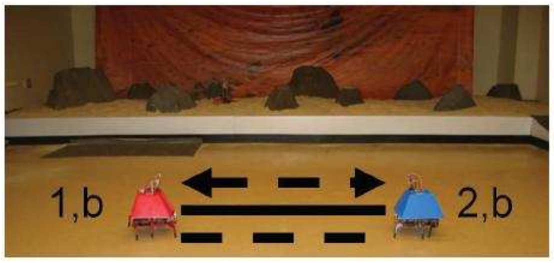
Three robots are configured with the defined EGG, as shown in Figure 3.10(a).We define the length of the triangle sides to be $d_b = d_c = 1$ m in order to easily photograph the execution. Each robot begins with the label a and begins performing the $goTo$ from control mode a. As in the two-robot EGG, two robots are relabeled b, and an edge is added to the network graph, shown in Figure 3.10(b). However, now a robot labeled a is available to



(a) Initial setup: communication edge exists



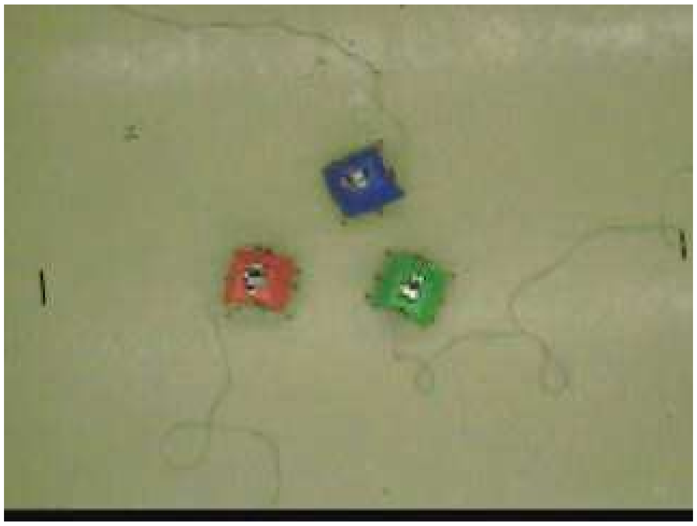
(b) Robot 2 can sense robot 1. A sense edge exists in the proximity graph, indicated by the dashed arrow.

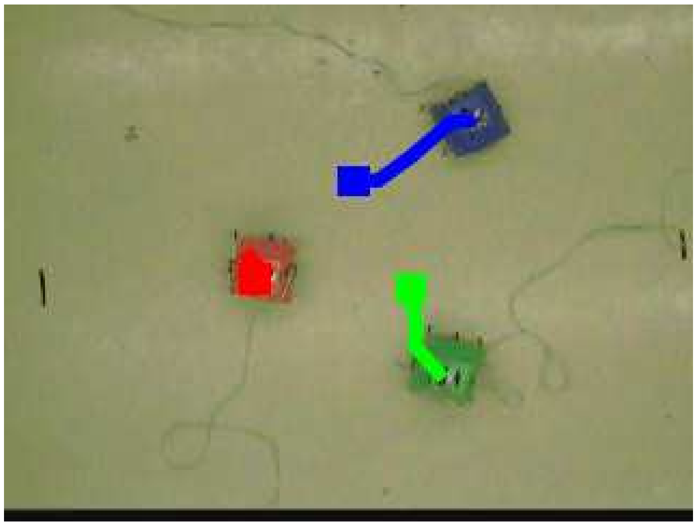


(c) The rule is applied, each robot is relabeled 'b', and an edge is added in the network graph, indicated by the solid line.

Figure 3.9. Two robot system performing "bar".

complete rule $r_2 \in \Phi$. Each robot is relabeled c (Figure 3.10(c)) , and two edges are added to the network graph. While the blue and green robots (top and right in Figure 3.10(c)) can perceive other robots, the red robot (left in Figure 3.10(c)) cannot. Therefore, it rotates left in the same spot, while the blue and green robots rotate and adjust their distances to each other and the red robot. By the time the red robot can perceive either of them, the robots have achieved a triangle with edges of approximate length $d_c = 1$ m. During all executions, the approximate triangle was formed (Figure 3.10(c)). Also, no inconsistent trajectories occurred.


(a) Initial setup.


(b) Red rotates left, looking for robots, while blue and green adjust.

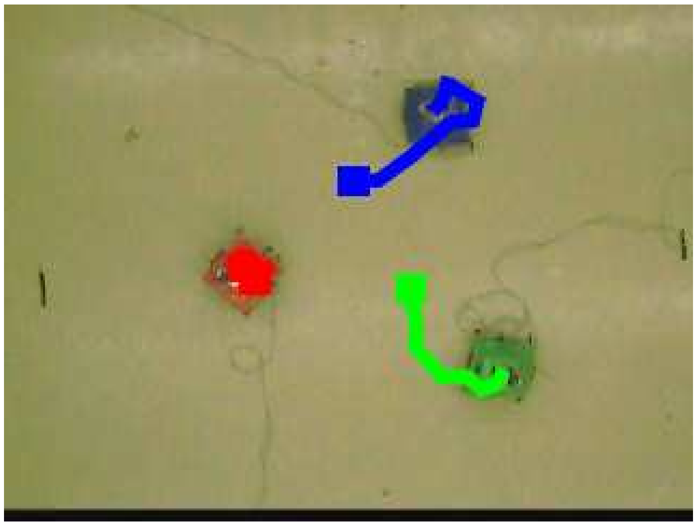

(c) Robots have formed equilateral triangle with sides of 1 m.

Figure 3.10. Three robot system performing triangle.

CHAPTER 4

RIGID AND PERSISTENT FEASIBILITY AND FORMATION GRAPH GENERATION

Chapter 3 presented methods and experiments for successfully implementing EGG systems with a small network of robots. In this chapter, we present methods for choosing a suitable network graph for the desired formation. We define a *formation graph* as a desired network graph for implementing the desired formation with the multi-robot network.

The number of potential formation graphs for the network grows exponentially with the network size. Therefore, we need an efficient way to choose a “good” network graph for a given desired formation.

We would like a network graph such that the network, when in formation, is a *rigid* network. However, since the network has a proximity range (the maximum range at which robots can sense/communicate with each other), this implies that network constraints that are longer than the proximity range cannot be maintained. It is unclear, under the sensing and communication limitations we consider, how a multi-robot network will maintain constraints that are longer than the proximity range. This suggests that certain formations are *feasible* with rigid and persistent network graphs, and certain formations are not. In this chapter, we present a method for determining if a desired formation is *rigidly* and *persistently feasible* with respect to the proximity range of the network.

In Chapter 4.1, we review the concept of rigidity. This includes a discussion of *rigid graphs*: network graphs such that the network is rigid for (practically) all network states. We also define the *rigid feasibility* of a desired formation. Given the proximity range of the network, a desired formation is rigidly feasible if, while in formation, the network graph can be a rigid graph whose edges are less than or equal to the proximity range. Otherwise, we say that the desired formation is *rigidly infeasible*. We also present an algorithm for efficiently determining the rigid feasibility of a desired formation for the network's proximity range. This is an efficient algorithm that generates a rigid network graph that respects the network's proximity range. Specifically, the generated network graph is a rigid graph with edge lengths less than or equal to the proximity range of the network.

Chapter 4.2 reviews *persistence* as it pertains to multi-robot networks. This also includes a discussion of *persistent graphs*: network graphs such that the network is persistent for (practically) all network states. Further, we define the *persistent feasibility* of a desired formation. Given the proximity range of the network, a desired formation is persistently feasible if, while in formation, the network graph can be a persistent graph whose edges are less than or equal to the proximity range. We show that, for a specific proximity range, all rigidly feasible formations are also persistently feasible. Also, the same algorithm for generating a rigid graph for the network also generates a persistent graph that respects the network's proximity range.

Graph operations for assembling persistent network graphs are presented in Chapter 4.3. For these operations, we start with an initial graph with two vertices and one edge. Each graph operation adds vertices and edges to the graph. Thus, a sequence of these graph operations generates a sequence of graphs such that the last graph is the desired, persistent network graph. Also, each intermediate graph in the sequence is a persistent graph. We present graph operations for assembling persistent network graphs that respect the proximity range of the network. We also present an algorithm for automatically generating a sequence of graph operations for a persistent formation graph.

In Chapter 4.4, we define *stably, persistently feasible* desired formations. These are persistently feasible formations that can be realized with the network using an acyclic, persistent network graph that respects the network's proximity range. Such acyclic, persistent graphs are called *stable, persistent graphs*, and facilitate the automatic definition of control laws for stabilizing the network to a desired formation. We present an algorithm for determining if a desired formation is stably, persistently feasible, given the proximity range of the network. We also present an algorithm for generating a sequence of graph operations for assembling a stable, persistent formation graph. Stable, persistent formation graphs and their corresponding sequences of graph operations are used for defining control laws for persistent networks (Chapter 5), as well as in implementing an EGG system for formation assembly under range constraints, presented in Chapter 6.

4.1 Rigid Feasibility and Rigid Graph Generation: The Modified "Pebble Game"

Here, we present rigid feasibility in terms of range constraints. First, we present rigidity as it has been defined in previous work. Then, we define rigid feasibility under range constraints and provide an algorithm for determining if a desired formation is rigidly feasible for the network.

4.1.1 Rigidity

Here, we examine the potential rigidity of a network given a desired formation $\overline{P}$ and a network graph G. Recall our multi-robot system model from Chapter 2, and consider a multi-robot network with network graph G that is initially *in formation* such that, $\forall(i, j) \in N \times N$, $\|x_i(0) - x_j(0)\| = \|\overline{p}_i - \overline{p}_j\|$. For a network initially in formation, an edge-consistent trajectory is a network trajectory such that, $\forall\left((v_i, v_j), t\right) \in E(G) \times T$, $\left\|x_i(t) - x_j(t)\right\| = \left\|\overline{p}_i - \overline{p}_j\right\|$. A rigid trajectory is a network trajectory such that, $\forall\,(i, j, t) \in N \times N \times T$, $\left\|x_i(t) - x_j(t)\right\| = \left\|\overline{p}_i - \overline{p}_j\right\|$ (i.e., the network stays in formation during the entire trajectory). For a given formation $\overline{P}$ and network graph G, the multi-robot network is rigid if and only if all edge-consistent trajectories of the network are also rigid trajectories. Otherwise, it is a flexible network. The rigidity of the network in a desired formation $\overline{P}$ with network graph G implies that the target formation can be maintained by guaranteeing that the constraints represented by $E(G)$ are maintained. Figure 4.1 gives examples of a flexible network, while Figure 4.2 gives an example of a rigid network.

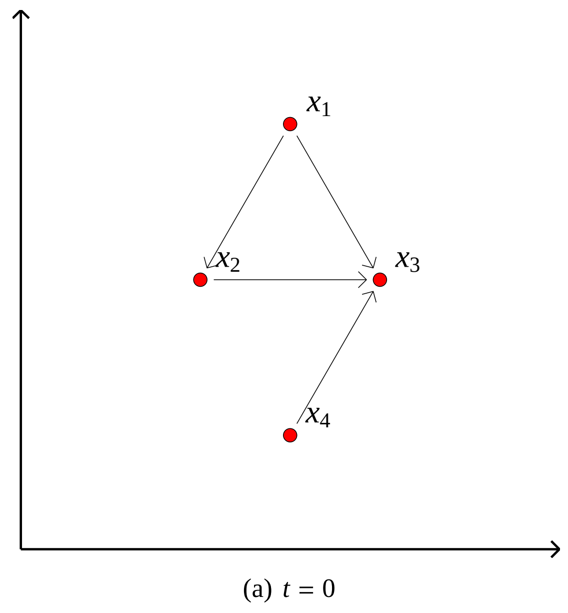


Figure 4.1. A flexible network. We assume that $t_1 > 0$. The line from x_4 represents circular motion that robot 4 can perform and still satisfy its constraint with robot 3. Robot 4 can move in a manner that changes its distance to robots 1 and 2.

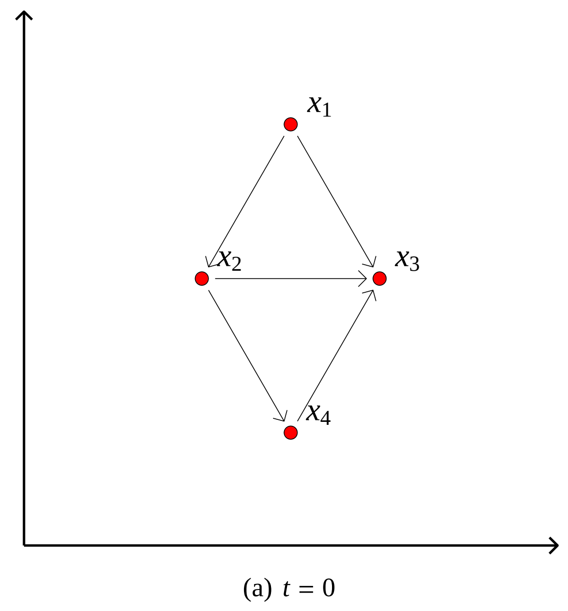


Figure 4.2. A rigid network. We assume that $t_1 > 0$. If all constraints are satisfied during continuous motion, then the network geometry does not change.

4.1.2 Infinitesimal Rigidity

Here, we review the concept of infinitesimal rigidity as presented in [37, 38, 39]. The infinitesimal rigidity of a network is a stronger condition than rigidity in that all infinitesimally rigid networks are rigid. While some rigid networks are *not* infinitesimally rigid, the infinitesimal rigidity of a network is a much easier condition to both test for and guarantee through our choice of network graph.

We assume that $x_i(t)$ is continuously differentiable $\forall i \in N$. Since we have defined an edge-consistent trajectory such that, $\forall (v_i, v_j) \in E(G)$, the distance between points $x_i(t)$ and $x_j(t)$ remains constant all along the trajectory, this implies that, $\forall \left((v_i, v_j), t\right) \in E(G) \times T$,

$$\frac{d}{dt}\left(\left\|x_i(t) - x_j(t)\right\|^2\right) = \left(x_i(t) - x_j(t)\right)^T \left(\dot{x}_i(t) - \dot{x}_j(t)\right) = 0. \tag{5}$$

Since we assume that the network is in formation, we can assume without loss of generality that $x_i(0) = \bar{p}_i \ \forall i \in N$. Under this assumption, $\forall i \in N$, the assignment of constant instantaneous velocities $u_i \in \mathbb{R}^2$ such that, $\forall (v_i, v_j) \in E(G)$, $\dot{x}_i(0) = u_i$ and $\dot{x}_j(0) = u_j$ satisfies (5) is described as an *infinitesimal motion* of the network [39]. Let $U \in \mathbb{R}^{2n}$ be defined by the infinitesimal motion such that $U = \left[u_1^T, \ldots, u_n^T\right]^T$. Then (5) is represented in matrix form $\forall (v_i, v_j) \in E(G)$ as

$$M(P, G)U = 0,$$

where $M(P, G)$ is known as the *rigidity matrix* [39]. The rigidity matrix has $|E(G)|$ rows and $2n$ columns. For each edge $(v_i, v_j) \in E(G)$, each row m_{ij} of $M(P, G)$ represents the equation for that edge as a 2n-vector of the form

$$m_{ij} = (0, \ldots, (\bar{p}_i - \bar{p}_j)^T, 0, \ldots, 0, (\bar{p}_j - \bar{p}_i)^T, \ldots, 0).$$

Here, $(\bar{p}_i - \bar{p}_j)^T$ is in two columns for vertex i, $(\bar{p}_j - \bar{p}_i)^T$ is in the columns for j, and zeroes are elsewhere [39]. A network with $n \geq 2$ points in $\mathbb{R}^2$ and in formation $\bar{P}$ is *infinitesimally rigid* if and only if $rank(M(P, G)) = 2n - 3$ [38, 39].

Infinitesimal rigidity implies rigidity, but rigidity does not imply infinitesimal rigidity

[37]. Still, the rigidity matrix is an effective way to demonstrate infinitesimal rigidity, and thus rigidity, based on the desired formation and network graph.

4.1.3 Generic Rigidity

The rigidity of a network depends both on the topology (G) and the state of the network ($X(t)$). A *generically rigid graph* is an network graph for which there exists a formation $\bar{P} \in \mathbb{R}^{2n}$ such that the network is infinitesimally rigid when in formation. Note that generic rigidity is a property of a network graph. Therefore, we refer to generically rigid graphs as *rigid graphs*. If the network graph G is rigid and the network is infinitesimally rigid while in formation $\bar{P} \in \mathbb{R}^{2n}$, we say that $\bar{P}$ is a *generic formation* of G.

If G is rigid, then the generic formations of G form a dense, open subset of $\mathbb{R}^{2n}$ [38]. This implies that, for any generically rigid graph G, any desired formation $\bar{P}'$ can be well-approximated by a generic formation $\bar{P}$ such that the network, when in formation, is infinitesimally rigid and, therefore, rigid.

4.1.4 Rigid Feasibility and Rigid Formation Graph Generation

The proximity range $\Delta \in \mathbb{R}^+$ of the network limits the maximum length of a constraint in the network. It is unclear, without further assumptions, how a pair of robots would maintain a distance greater than Δ from each other, since they would be outside their sensor/communication range. Given a desired formation $\bar{P}$, we want to determine if, while in formation, the network can have a rigid network graph such that all edges of the network graph are less than or equal to the proximity range. To this end, we define *rigid feasibility* as follows:

Definition 4.1 *For a multi-robot network with $n \in \mathbb{N} : n \geq 2$ members and proximity range $\Delta \in \mathbb{R}^+$, a desired formation $\bar{P} \in \mathbb{R}^{2n}$ is rigidly feasible if and only if there exists a network graph G^Δ such that G^Δ is rigid and, $\forall (v_i, v_j) \in E(G^\Delta)$, $\left\| \bar{p}_i - \bar{p}_j \right\| \leq \Delta$.*

Adding edges to a rigid graph cannot cause it to lose rigidity (i.e., it will stay rigid). A *minimally rigid graph* is rigid but does not remain rigid after the removal of any single edge. By Laman's theorem [43], a network with robots defined in $\mathbb{R}^2$ with $n \geq 2$ vertices 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* [44]. The pebble game algorithm constructs minimally rigid graphs, with a worst case performance of $O(n^2)$ [44]. 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 edge can be covered by a pebble from a vertex incident to that edge. To keep track of pebbles, the pebble game works with a directed graph, where a directed edge $(v_i, v_j) \in E(G)$ indicates that edge (v_i, v_j) is covered by a pebble from vertex $v_i \in V(G)$. For a given $v \in V(G)$, the pebbles of v can only cover edges whose tail is v.

The pebble game starts with a directed graph with no edges and attempts to add each potential edge one at a time to the pebble covering in a manner that ensures the second part of Laman's theorem 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 the graph. If part 2 of Laman's theorem is satisfied, we say that a *valid pebble covering* has been found. If a valid pebble covering of $2n - 3$ such edges is found, then this implies that the first part of Laman's theorem is satisfied and the graph is minimally rigid. For more detail on the implementation of this algorithm, see [44].

To test for a minimally rigid graph that satisfies Definition 4.1, we modify the pebble game algorithm so that it only considers edges of length less than or equal to Δ. The modified pebble game is described in Algorithm 1.

The following Theorem 4.2 states the effectiveness of the modified pebble game to test for rigid feasibility.

Algorithm 1 $ModifiedPebbleGame(\overline{P}, \Delta)$

```
Require: $\overline{P} \in \mathbb{R}^{2n}$ is a formation of $n$ positions such that, $\forall i \in \{1, \ldots, n\}$, $\overline{p_i} \in \mathbb{R}^2$
Require: $\Delta \in \mathbb{R}^+$ is the proximity range of the network
  Initialize network $G^\Delta$such that $V(G^\Delta) := \{v_1, \ldots, v_n\}$, $E(G^\Delta) := \emptyset$
  Initialize $rigid$ := false
  for all possible edges $e = (v_i, v_j) \in V(G^\Delta) \times V(G^\Delta)$ such that $v_i \neq v_j$ and $\left\|\overline{p_i} - \overline{p_j}\right\| \leq \Delta$,
  and while $rigid$ = false do
    $E(G^\Delta) := E(G^\Delta) \cup e$;
    Rearrange edge directions to try to find a valid pebble covering;
    if a valid pebble covering is not found then
      $E(G^\Delta) := E(G^\Delta) \setminus e$;
    end if
    if $|E(G^\Delta)| = 2n - 3$ then
      $rigid$ := true;
    end if
  end for
  return $(rigid, G^\Delta)$;
```

Theorem 4.2 *For a multi-robot network with $n \in \mathbb{N} : n \geq 2$ members and proximity range $\Delta \in \mathbb{R}^+$, a desired formation $\overline{P} \in \mathbb{R}^{2n}$ is rigidly feasible if and only if the algorithm $ModifiedPebbleGame(\overline{P}, \Delta)$ returns a minimally rigid graph.*

Proof: Definition 4.1 is satisfied for formation $\overline{P}$ only if there exists a rigid graph G^Δ such that, $\forall (v_i, v_j) \in E(G^\Delta)$, $\left\|\overline{p_i} - \overline{p_j}\right\| \leq \Delta$. This implies the existence of a minimally rigid graph with the same properties. Assume that G^Δ exists (with all edges of length less than or equal to Δ), but that $ModifiedPebbleGame(\overline{P}, \Delta)$ fails to return a minimally rigid graph. Note from [44] that the unmodified pebble game always generates a rigid graph, and does so by considering each edge and adding it to a flexible graph until it becomes minimally rigid. Therefore, the failure of $ModifiedPebbleGame(\overline{P}, \Delta)$ implies that no such graph can be generated considering only edges such that their distance in the network would be less than or equal to Δ. Since the unmodified pebble game always returns a minimally rigid graph [44], this implies that, for any rigid graph G, $\exists (v_i, v_j) \in E(G) : \left\|\overline{p_i} - \overline{p_j}\right\| > \Delta$. However, this violates our assumption that G^Δ exists. Therefore, the formation is rigidly feasible only if $ModfiedPebbleGame(\overline{P}, \Delta)$ returns a minimally rigid graph.

When the modified pebble game produces a minimally rigid graph G^{Δ} such that, $\forall (v_i, v_j) \in E(G^{\Delta})$, $\left\|\bar{p}_i - \bar{p}_j\right\| \leq \Delta$, then the conditions of Definition 4.1 are satisfied. ■

4.2 Persistent Feasibility and Persistent Formation Graph Generation

Here, we present persistent feasibility in terms of range constraints. First, we present persistence as it has been defined in previous work. Then, we define persistent feasibility under range constraints. Further, we demonstrate that rigid feasibility and persistent feasibility are equivalent. We also show that the modified pebble game algorithm generates minimally persistent graphs.

4.2.1 Persistence

To discuss persistence, we must first present *constraint consistence*. Informally, we say that constraint consistence means that all constraints are satisfied as long as all robots satisfy their individual constraints, i.e., no subset of robots can satisfy their constraints in a manner which prevents another robot from satisfying a constraint. Constraint consistence is determined by the number and orientation of the constraints. Figure 4.3 shows a constraint inconsistent network, while Figure 4.4 shows a constraint consistent network. For a more rigorous definition, see [40]. A network is persistent if and only if it is rigid and constraint consistent [40], as in Figure 4.4.

We say that a graph is *generically constraint consistent* if all of its vertices have an out-degree less than or equal to two [40]. Thus, we refer to generically constraint consistent graphs as *constraint consistent graphs*.

Similar to generic rigidity, we say that a graph is *generically persistent* if it is generically rigid and generically constraint consistent. Like generic rigidity, generic persistence applies to graphs, not networks. 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 (i.e., it is constraint consistent, but no edge can be removed without losing rigidity) [40].

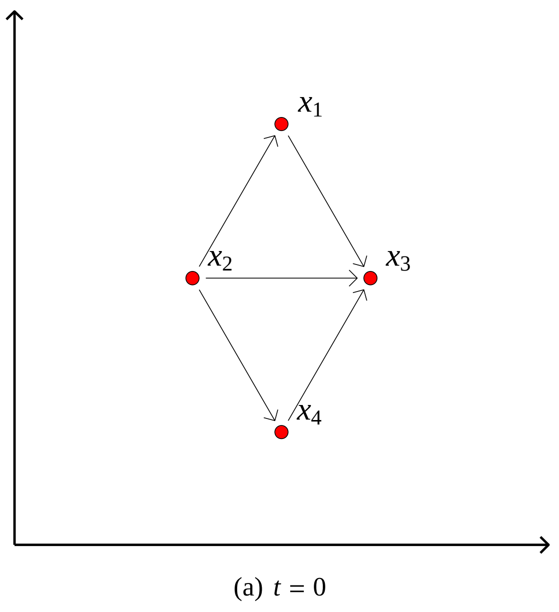


Figure 4.3. A constraint inconsistent network. We assume that $t_1 > 0$. Here, robot 4 can perform circular motion around robot 3. If robot 4 moves, robot 2 cannot move in a way that preserves the distances between robot 2 and robots 1, 3, and 4.

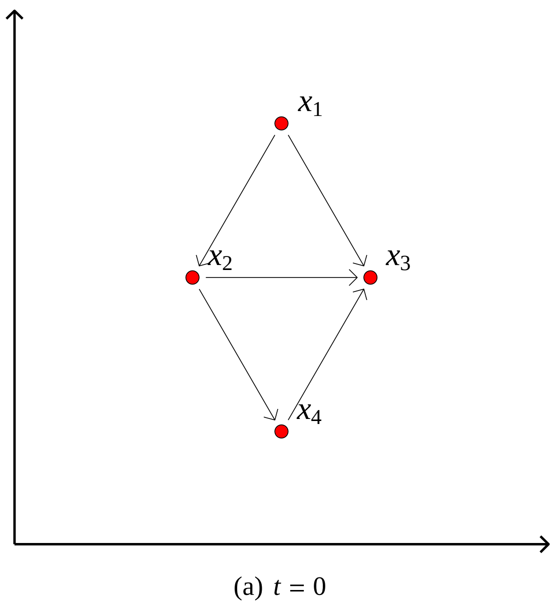


Figure 4.4. A persistent network. The network graph is rigid and constraint consistent. We assume that $t_1 > 0$. If robot 4 satisfies its constraint, the other robots maintain formation during continuous motion.

4.2.2 Rigidity, Constraint Consistence, and Persistence Summary

To summarize the topological notions of rigidity, constraint consistence, and persistence and their relations, see Table 2. Rigidity tells us whether or not we have sufficient edges in our graph to guarantee that the formation is maintained by only maintaining its edge lengths. Therefore, in Table 2, Graph 1 is flexible (i.e. not rigid), while Graphs 2 and 3 are rigid. Note that rigidity does not depend on the orientation of the edges.

Unlike rigidity, constraint consistence *does* depend on the orientation of the edges. While Graph 1 in Table 2 is flexible, it is constraint consistent, since all vertices have an out-degree less than or equal to two. This implies that the robots can maintain these edges regardless of how robots 2 and 4 satisfy their constraints with robot 3. Still, the formation may deform, since the graph is flexible. On the other hand, Graph 2 is rigid, but not constraint consistent. While the maintenance of the edges would preserve the formation, it is possible for robot 4 to satisfy its constraint with robot 3 such that robot 2 cannot satisfy all of its constraints, as in Figure 4.4.

Of the graphs in Table 2, only Graph 3 is both rigid and constraint consistent, which makes it the only persistent graph example. The maintenance of the edges ensures that the formation does not deform, and all robots can, in fact, maintain these edges during any continuous motion.

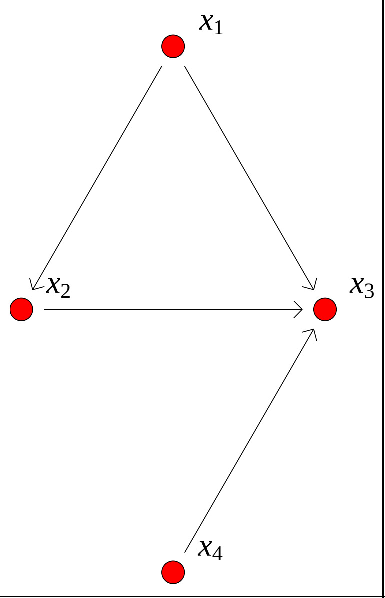
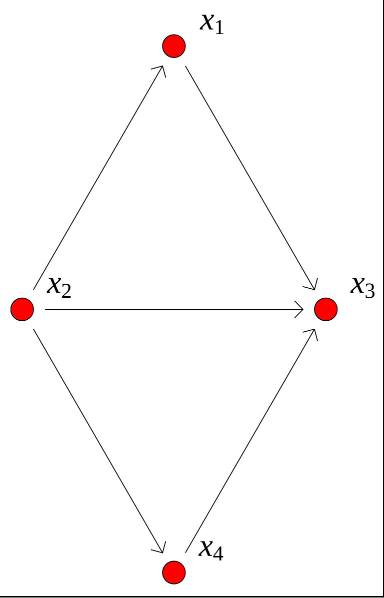
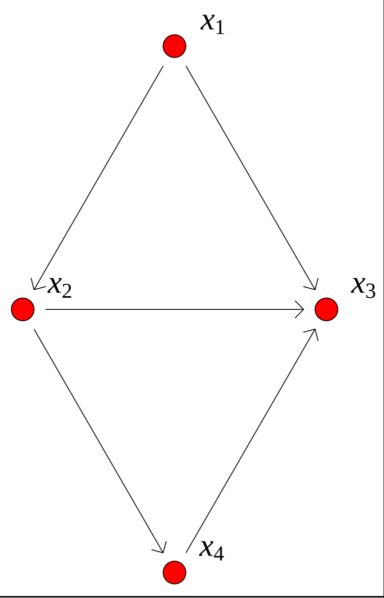
4.2.3 Persistent Feasibility

We define *Persistent feasibility* as follows:

Definition 4.3 *For a multi-robot network with $n \in \mathbb{N} : n \geq 2$ members and proximity range $\Delta \in \mathbb{R}^+$, a desired formation $\bar{P} \in \mathbb{R}^{2n}$ is* persistently feasible *if and only if a exists a network graph G^Δ such that G^Δ is persistent and, $\forall (v_i, v_j) \in E(G^\Delta)$, $\left\|\bar{p}_i - \bar{p}_j\right\| \leq \Delta$.*

For any minimally rigid graph, it is possible to assign directions to the edges such that the obtained directed graph is minimally persistent [23]. Therefore, we have the following Theorem 4.4 describing necessary and sufficient conditions for a target formation to be

Table 2. Rigidity, Constraint Consistence, and Persistence Examples Table

Graph 1	Graph 2	Graph 3
		
Flexible	Rigid	
Constraint Consistent	Constraint Inconsistent	Constraint Consistent
Not Persistent	Not Persistent	*Persistent*

persistently feasible.

Theorem 4.4 *For a multi-robot network with proximity range $\Delta \in \mathbb{R}^+$, a desired formation $\bar{P} \in \mathbb{R}^2$ is persistently feasible if and only if it is rigidly feasible.*

Proof: If $\bar{P}$ is rigidly feasible, then, by Definition 4.1, there exists a rigid graph G^Δ that is rigid and, $\forall (v_i, v_j) \in E(G^\Delta)$, $\left\|\bar{p}_i - \bar{p}_j\right\| \leq \Delta$. This implies the existence of a minimally rigid graph such that, $\forall (v_i, v_j) \in E(G^\Delta)$, $\left\|\bar{p}_i - \bar{p}_j\right\| \leq \Delta$. The directions of the edges of this minimally rigid graph can be assigned such that it is a minimally persistent graph [23], implying that Definition 4.3 is satisfied and the formation is persistently feasible. Since a graph is persistent if and only if it is rigid and constraint consistent, then $\bar{P}$ is not persistently feasible if it is not rigidly feasible. ∎

Theorem 4.4 shows that the modified pebble game tests for both rigid and persistent feasibility.

4.2.4 Persistent Graph Generation

Here, we show that the pebble game algorithm also generates minimally persistent graphs.

A graph is minimally persistent if and only if it is minimally rigid and no vertex has an out-degree larger than two [40]. We denote the out-degree of a vertex v by $deg^{-}(v)$. Note that the pebble game produces a directed graph G^{Δ}, where each edge $(v_i, v_j) \in E(G^{\Delta})$ is covered by one of two pebbles from vertex $v_i \in V(G^{\Delta})$. Thus, we have the following theorem:

Theorem 4.5 *The pebble game and modified pebble game algorithms generate minimally persistent graphs.*

Proof: Assume that G^{Δ} is a rigid graph successfully generated by the pebble game. In [44], it is shown that the pebble game generates a minimally rigid graph. Since each directed edge $(v_i, v_j) \in E(G^{\Delta})$ represents the edge being covered by one of two pebbles from vertex $v_i \in V(G^{\Delta})$, this implies that, $\forall v \in V(G^{\Delta})$, $deg^{-}(v) \leq 2$. This implies that G^{Δ} is constraint consistent. Since G^{Δ} is constraint consistent and minimally rigid, it is also minimally persistent. This also holds for the modified pebble game. ■

4.3 Persistent Graph Operations

Given a minimally persistent formation graph G for the network, a *Henneberg sequence* [45] of *graph operations* can be defined for assembling it. These sequences start out with an initial, *leader-first-follower graph* G_2 such that $V(G_2)$ has two vertices and $E(G_2)$ has one edge between them. If there are more than two vertices in $V(G)$, the first graph operation in the sequence adds a vertex and two edges to the G_2, producing a new graph: G_3. The next graph operation adds vertices and edges to G_3. In this manner, a sequence of graph operations and an initial graph G_2 defines a sequence of graphs $(G_n) = (G_2, \ldots, G_n)$ such that $G_n = G$, the final graph in the sequence. Furthermore, each intermediate graph G_i : $2 \leq i \leq n$ is also minimally persistent.

Here, we describe traditional graph operations. We introduce new graph operations for assembling minimally persistent graphs that respect the proximity range of the network. We also present an algorithm for automatically generating sequences of these graph operations for assembling a given, minimally persistent graph.

4.3.1 Leader-First-Follower Pairs

For a minimally persistent graph G, we define a *leader-first-follower pair* as a pair of robots corresponding to adjacent vertices $(v_l, v_f) \in V(G) \times V(G)$ such that $deg^{-}(v_l) = 0$, $deg^{-}(v_f) = 1$, and $\exists(v_f, v_l) \in E(G)$. We say that vertex l is the *leader*, and f is the *first-follower*. In such a network, all remaining robots are simply called *followers* (all robots i such that $i \in N \setminus \{l, f\}$).

The leader robot has no constraints, and thus has two degrees of freedom, implying that the persistent formation will follow the leader robot in $\mathbb{R}^2$. Similarly, the follower robot has one constraint, and thus one degree of freedom, implying that the persistent formation will rotate around the leader robot as the follower robot performs circular motion around the leader. Thus, the leader and first-follower establish the position of the formation in the environment, in terms of both translation and rotation. For a persistent graph, edge-reversing operations can make any pair of adjacent robots a leader-first-follower pair with the graph remaining persistent [23]. A leader-first-follower pair is demonstrated in Figure 4.4 with robots 3 and 4.

4.3.2 Traditional Persistent Graph Operations

In a multi-robot network, achieving a persistent formation requires robots with (initially) no constraints to interact and establish constraints. Such a sequence of robots interactions, if successful, results in a persistent formation, with inter-robot distances corresponding to the target formation.

Graph operations are used to represent such a sequence of robot interactions. In [23], graph operations are presented for assembling and modifying persistent graphs. Initially,

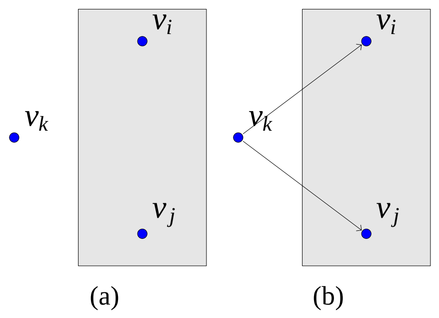


Figure 4.5. A vertex addition operation. In this figure, the shaded area represents a minimally persistent graph before the operation. The resulting graph is always minimally persistent, as well.

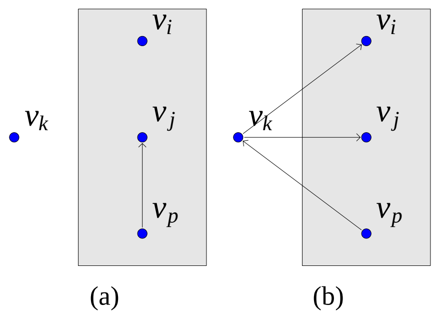


Figure 4.6. An edge-splitting operation. In this figure, the shaded area represents a minimally persistent graph before the operation. The resulting graph is always minimally persistent.

a leader-first-follower seed graph is formed, with leader robot l, first-follower robot f, and graph G_2 such that $V(G_2) = \{v_l, v_f\}$, and $E(G_2) = \{(v_f, v_l)\}$.

In [23], directed vertex addition and edge-splitting operations are presented. Consider a graph G_i in a Henneberg sequence such that $2 \leq i \leq n$, $\{v_i, v_j, v_p\} \subseteq V(G_i)$, $(v_p, v_j) \in E(G_i)$, and $v_k \notin V(G_i)$. A vertex addition consists of adding v_k to $V(G_i)$ and adding edges $\left\{(v_k, v_i), (v_k, v_j)\right\}$ to $E(G_i)$, producing the next graph in the sequence, G_{i+1}, with one new vertex and two new edges. Figure 4.5 shows a vertex addition operation.

An edge-splitting operation consists of adding v_k to $V(G_i)$ and adding edges $\left\{(v_k, v_i), (v_k, v_j)\right\}$ to $E(G_i)$, while also removing edge (v_p, v_j) from $E(G_i)$, producing the next graph, G_{i+1}. Figure 4.6 shows an edge-splitting operation.

Any minimally persistent graph can be assembled from a leader-first-follower seed graph and a sequence of vertex additions and edge-splitting operations, along with edge and path reversing operations [23]. Additionally, each intermediate graph is persistent

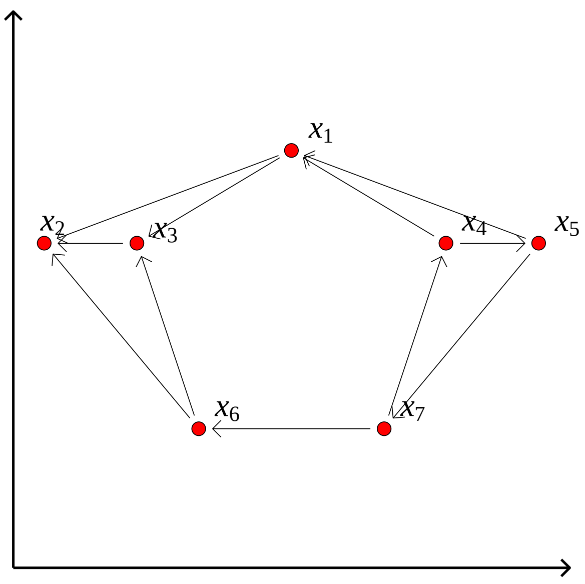


Figure 4.7. An example network where performing an inverse edge-splitting operation introduces a new edge whose length is greater than all pre-existing edges. This new edge could violate the proximity range of the network.

[23]. These sequences of graph operations to build a desired graph are typically defined by performing inverse graph operations on the desired graph, defining a reverse sequence of graphs until the last graph is a leader-first-follower seed graph: $(G_n, \ldots, G_2)$. However, these methods are completely graph based, and do not take into account the proximity range for the multi-robot network. Consider Figure 4.7. This network has a minimally rigid, persistent graph. An inverse vertex addition cannot be performed. Also, note that any inverse edge-splitting operation will introduce a new edge into the network which has a length longer than any other edge. This new edge could violate the proximity range of the network. *Therefore, given a formation and a proximity range limit on the edge lengths of a network, certain network graphs cannot be assembled by these traditional operations without introducing a constraint that violates the proximity range.*

4.3.3 Persistent-Δ Operations

To construct persistent graphs under proximity range constraints, we present two new graph operations. These, combined with traditional vertex addition, allow any persistent graph

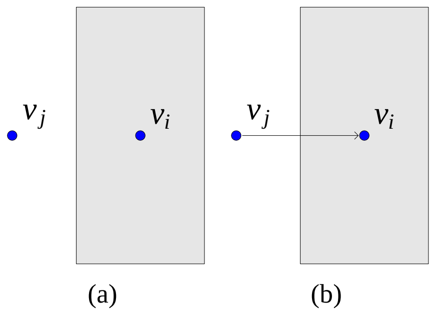


Figure 4.8. A single-vertex addition. The shaded area represents a minimally persistent graph before the operation. A single vertex with an edge is added to graph G_k, producing the next graph in the sequence: G_{k+1}.

with a leader-first-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 $V(op) = V$ is a set of vertices to add to the graph, and $E(op) = E$ is a set of edges to add to the graph.

A *vertex addition* is a persistent-Δ operation defined as in Chapter 4.3.2. A vertex addition is represented as $vertexAddition(v_i, v_j, v_k) = (\{v_k\}, \{(v_k, v_i), (v_k, v_j)\})$.

Consider a directed graph G_k such that $2 \leq k \leq n$, $v_i \in V(G)$, $v_j \notin V(G)$. *Single-vertex addition* consists of adding a vertex v_j to $V(G_k)$ and adding edge (v_j, v_i) to $E(G_k)$, producing the next graph G_{k+1} in the sequence. A single-vertex addition is represented as $singleVertex(v_i, v_j) = (\{v_j\}, \{(v_j, v_i)\})$. 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. Figure 4.8 shows a single-vertex addition.

Consider a directed graph G_k such that $2 \leq k \leq n$, $(v_i, v_j) \in V(G_k) \times V(G_k)$ and $(v_j, v_i) \notin E(G_k)$. *Edge insertion* consists of adding edge (v_j, v_i) to $E(G_k)$, producing the next graph G_{k+1} in the sequence. An edge insertion is represented as $edgeInsertion(v_i, v_j) = (\emptyset, \{(v_j, v_i)\})$. Figure 4.9 shows an edge insertion operation.

4.3.4 Persistent-Δ Sequence Generation

Here, we describe how persistent-Δ operations can be used to construct any persistent graph with a leader-first-follower pair.

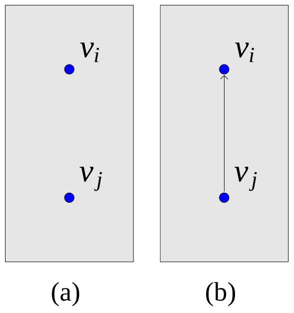


Figure 4.9. An edge insertion operation. As before, the shaded area represents a minimally persistent graph before the operation. A single edge is added to graph G_k, producing the next graph in the sequence: G_{k+1}.

If G is a minimally persistent graph and $\exists(v_i, v_j) \in V(G) \times V(G)$ such that $deg^-(v_i) \geq 1$ and vertex $deg^-(v_j) \leq 1$, then there is a directed path from v_i to v_j [23]. If $(v_l, v_f) \in V(G) \times V(G)$ are a leader-first-follower pair, respectively, then $\forall v \in V(G) \setminus \{v_l, v_f\}$, $deg^-(v) = 2$ [40]. This leads to the following lemma:

Lemma 4.6 *Let G be a minimally persistent graph such that $v_l \in V(G)$ is the vertex of the leader and $v_f \in V(G)$ is the vertex of the first-follower in a leader-first-follower pair. This implies the existence of a directed path from all vertices $v \in V(G) \setminus v_l$ to v_l.*

Proof: Since robots l and f are a leader-first-follower pair, this implies that there exists a directed path from v_f to v_l and that $deg^-(v_l) < deg^-(v_f) \leq 1$. This implies that, $\forall v \in V(G) \setminus \{v_l, v_f\}$, $deg^-(v) = 2$. Then there is a directed path from v to v_f and v_l. This implies that there exists path from all vertices in $V(G) \setminus v_l$ to v_l. ∎

This leads us to an algorithm for constructing a sequence of graph operations to construct a minimally persistent graph. We define a *leader-first-follower seed* as a graph G_2 such that $V(G_2) = \{v_l, v_f\}$ and $E(G_2) = \{(v_f, v_l)\}$. Here, vertex v_l is the leader vertex, and vertex v_f is the follower vertex.

Any minimally persistent graph can be constructed from a leader-first-follower seed by a sequence of persistent-Δ graph operations. First, given a minimally persistent formation graph G^Δ, a graph G is initialized to the leader-first-follower seed such that $G = G_2$ using

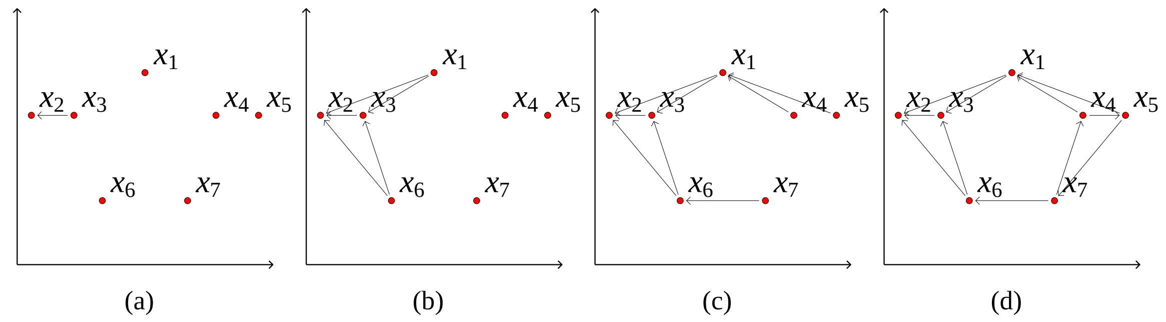


Figure 4.10. A sequence of Persistent-Δ operations constructing a framework. 4.10(a): The initial leader-first-follower seed. 4.10(b): Two vertex additions are performed. 4.10(c): No more vertex additions are possible. Three single-vertex additions are performed. 4.10(d): Three edge insertions are performed, one for each single-vertex addition.

the leader and follower vertices in G^{Δ}. Until all vertices and edges of G^{Δ} are present in G, 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 G. This process is repeated until all vertices and edges have been added to the graph, implying that $G = G^{\Delta}$. Algorithm 2 describes this process. In Algorithm 2, we represent concatenating element s to the end of sequence S by $S \cdot s$.

Figure 4.10 shows a resulting sequence of this algorithm. We have the following theorem for the effectiveness of this method:

Theorem 4.7 *For a minimally persistent graph G^{Δ} with a leader-first-follower pair, the persistent-Δ generation algorithm will generate a sequence of graph operations that construct G^{Δ} from a leader-first-follower seed.*

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

```
Require: Graph $G^{\Delta}$ exists such that $G^{\Delta}$ is minimally persistent with leader-first-follower
  pair $(v_l, v_f)$;
  Initialize $G_2$ such that $V(G_2) = \{v_l, v_f\}$ and $E(G_2) = \{(v_f, v_l)\}$;
  $G := G_2$;
  Initialize $S$ such that $S$ is a sequence of zero graph operations;
  while $|V(G)| < |V(G^{\Delta})|$ or $|E(G)| < |E(G^{\Delta})|$ do
    Initialize set of graph operations $s := \emptyset$;
    for all $(v_i, v_j) \in V(G) \times V(G)$ do
      {Generate all possible edge insertions}
      if $(v_j, v_i) \in E(G_n^{\Delta})$ and $(v_j, v_i) \notin E(G)$ then
        $s := s \cup edgeInsertion(v_i, v_j)$;
      end if
    end for
    $vertexAdded$ := false;
    for all $v_k \in V(G^{\Delta})$ such that $v_k \notin V(G)$ do
      {Generate all possible vertex additions}
      if $\exists (v_i, v_j) \in V(G) \times V(G)$ such that $v_i \neq v_j$ and $\{(v_k, v_i), (v_k, v_j)\} \in E(G^{\Delta})$ then
        $s := s \cup vertexAddition(v_i, v_j, v_k)$;
        $vertexAdded$ := true;
      end if
    end for
    if $vertexAdded$ = false then
      for all $v_j \in V(G^{\Delta})$ such that $v_j \notin V(G)$ do
        {Generate all possible single-vertex additions}
        if $\exists v_i \in V(G)$ such that $(v_j, v_i) \in E(G_n^{\Delta})$ then
          $s := s \cup singleVertex(v_i, v_j)$;
        end if
      end for
    end if
    for all operations $op \in s$ do
      {Perform all determined graph operations}
      $V(G) := V(G) \cup V(op)$;
      $E(G) := E(G) \cup E(op)$;
      $S := S \cdot op$;
    end for
  end while
  return $(G_2, S)$;
```

Proof: Assume that G^{Δ} exists, with (l, f) as the leader and first-follower of a leader-first-follower pair, and that G is the initialized leader-first-follower seed such that $G = G_2$. If G^{Δ} has only two vertices, the graph is constructed.

If there are more than two vertices, then, by Lemma 4.6, there exists a path from all vertices in $V(G^{\Delta}) \setminus \{v_l\}$ to vertex v_l. This implies that there exists a pair of vertices (v_i, v_j) such that $v_j \in V(G^{\Delta}), v_j \notin V(G)$, $v_i \in V(G)$, and $(v_j, v_i) \in E(G^{\Delta})$. 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 $V(G)$ and $E(G)$. Note that G always has the leader-first-follower pair. Therefore, if there are remaining vertices $v \in V(G^{\Delta})$ such that $v \notin V(G)$, then Lemma 4.6 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 $v \in V(G^{\Delta})$ such that $v \notin V(G)$. Since we have not added any vertices $v \notin V(G^{\Delta})$ to $V(G)$, this implies that, at this point, $V(G^{\Delta}) = V(G)$.

For all edges $(v_j, v_i) \in E(G^{\Delta})$, either $(v_j, v_i) = (v_f, v_l)$,the leader-first-follower's edge, or it does not. If (v_j, v_i) is the leader-first-follower's edge, then it was added to $E(G)$ when the leader-first-follower seed G_2 was initialized. If it is not the leader-first-follower edge, note that we have already proven that all vertices $V(G^{\Delta})$ are added to $V(G)$ such that $V(G^{\Delta}) = V(G)$. This implies that, for any remaining edges not added by vertex or single-vertex additions, there exists a pair of vertices $(v_i, v_j) \in V(G)$ such that $(v_j, v_i) \in E(G^{\Delta})$ and $(v_j, v_i) \notin E(G)$. These edges are added by edge insertions.

Since the algorithm uses these conditions to search for single-vertex additions and edge-insertions, all such operations are performed, guaranteeing that $V(G^{\Delta}) = V(G)$ and $E(G^{\Delta}) = E(G)$. ∎

4.4 Stably, Persistently Feasible Formations

In Chapter 4.2, we describe how to determine if desired formations were persistently feasible given the proximity range of the network. While all rigidly feasible formations are also persistently feasible, some rigidly feasible formations require minimally persistent graphs that contain *cycles*. Cyclic formation graphs can result in instabilities which depend on the initial state of the network, and are difficult to identify before their implementation on the network [15, 16].

Directed graphs that are minimally persistent and acyclic have been called *stably rigid graphs* in other works (see, for example, [16, 20]). A graph is stably rigid if and only if it can be assembled from a Henneberg sequence composed entirely of vertex additions [20]. This implies that these graphs are also minimally persistent [23]. Therefore, we call these graphs *stable, persistent graphs*.

It has been shown that not all persistently feasible formations can be assembled with only vertex addition operations [4]. However, all persistently feasible formations that cannot be assembled by vertex additions must contain cycles. Therefore, we present a method to determine if a persistently feasible formation can be assembled only with vertex additions. We define stable, persistently feasible formations as follows:

Definition 4.8 *For a multi-robot network with $n \in \mathbb{N} : n \geq 2$ members and proximity range $\Delta \in \mathbb{R}^+$, a desired formation $\bar{P} \in \mathbb{R}^{2n}$ is stably, persistently feasible if and only if there exists a network graph G^Δ such that G^Δ is persistent and acyclic and, $\forall (v_i, v_j) \in E(G^\Delta)$, $\left\|\bar{p}_i - \bar{p}_j\right\| < \Delta$.*

To determine if a formation described by $\bar{P}$ is stably, persistently feasible, we present the $StablePersistentDelta$ algorithm, shown in Algorithm 3. This is a polynomial time algorithm that attempts to assemble such a graph using edge weights defined by $\bar{P}$. For each potential pair of leader and first-follower robots, the algorithm attempts to build a minimally persistent graph using only vertex additions and edges with lengths less than Δ. If successful, this algorithm also defines a leader-first-follower seed graph and a sequence of vertex addition operations for assembling an acyclic, minimally persistent graph. This algorithm has a worst-case performance of $O(n^4)$.

Theorem 4.9 *A desired formation $\bar{P}$ is* stably, persistently feasible *for a multi-robot network with proximity range $\Delta \in \mathbb{R}^+$ if and only if the $StablePersistentDelta$ algorithm successfully returns a minimally persistent graph.*

Proof: From Definition 4.8 and [20], a desired formation $\bar{P}$ is stably, persistently feasible if and only if there exists a minimally persistent network graph G^Δ such that G^Δ can be assembled entirely from directed vertex additions, and, $\forall(v_i, v_j) \in E(G^\Delta)$, $\|\bar{p}_i - \bar{p}_j\| < \Delta$. From Algorithm 3, the $StablyRigidDelta$ algorithm first finds a pair of positions within Δ to be the leader and first-follower pair. Then, it attempts to find vertex additions to add the remaining vertices to the graph using vertex additions with edges less than Δ. The final graph G^Δ is only returned when all n vertices have been added, implying that G^Δ is, indeed, a minimally persistent graph assembled totally from vertex additions. Thus, $\bar{P}$ is stably, persistently feasible if the $StablePersistentDelta$ algorithm is successful.

To show that this is necessary, assume that, for our given proximity range Δ, G^Δ exists such that it is a stable, minimally persistent graph, but the $StablePersistentDelta$ algorithm fails to return such a graph. This implies that a leader and first-follower pair cannot be chosen such that a sequence of vertex addition operations produces a minimally persistent graph whose edges all have a length less than Δ. However, this contradicts the assumption that G^Δ exists and is a stable, minimally persistent graph. Therefore, $\bar{P}$ is stably, persistently feasible only if the $StablePersistentDelta$ algorithm successfully returns a minimally persistent graph. ■

The $StablePersistentDelta$ algorithm is used to automatically generate stable, persistent formation graphs for the network. In subsequent chapters in this work, given a desired formation $\bar{P}$ and the formation graph G^Δ generated by the $StablePersistentDelta$ algorithm,

Algorithm 3 $StablePersistentDelta(\overline{P}, \Delta)$

```
Require: $n \in \mathbb{N} : n \geq 2$ is the network size;
Require: $\overline{P} \in \mathbb{R}^{2n}$ is a formation of $n$ positions such that, $\forall i \in \{1, \ldots, n\}$, $\overline{p_i} \in \mathbb{R}^2$;
Require: $\Delta \in \mathbb{R}^+$;
  Initialize graph $G^\Delta$ such that $V(G^\Delta) = \emptyset$ and $E(G^\Delta) = \emptyset$;
  Initialize graph $G_2$ such that $V(G_2) = \emptyset$ and $E(G_2) = \emptyset$;
  for all $a \in N$ do
    $N_{add} := \{a\}$ ;
    $V_{add} := \{v_a\}$ ;
    $V(G^\Delta) := \emptyset$ ;
    $E(G^\Delta) := \emptyset$ ;
    for all $b \in N \setminus \{a\}$ do
      if $||\overline{p_a} - \overline{p_b}|| < \Delta$ ; then
        $N_{add} := N_{add} \cup \{b\}$ ;
        $V_{add} := V_{add} \cup \{v_b\}$ ;
        $E_{add} := \{(v_b, v_a)\}$;
        $V(G_2) := \{v_a, v_b\}$;
        $E(G_2) := \{(v_b, v_a)\}$;
        Initialize $S$ such that $S$ is a sequence of zero graph operations;
        while $|V(G^\Delta)| < n$ and $V_{add} \neq \emptyset$ do
          $V(G^\Delta) := V(G^\Delta) \cup V_{add}$ ;
          $E(G^\Delta) := E(G^\Delta) \cup E_{add}$ ;
          $V_{add} := \emptyset$ ;
          $E_{add} := \emptyset$ ;
          for all $k \in N \setminus N_{add}$ ; do
            if $\exists(v_i, v_j) \in V(G^\Delta) \times V(G^\Delta)$ such that $v_i \neq v_j$, $||\overline{p_k} - \overline{p_i}|| < \Delta$, and $||\overline{p_k} - \overline{p_j}|| <$
            $\Delta$ then
              $N_{add} := N_{add} \cup k$ ;
              $V_{add} := V_{add} \cup \{v_k\}$ ;
              $E_{add} := E_{add} \cup \{(v_k, v_i), (v_k, v_j)\}$ ;
              $S := S \cdot vertexAddition(v_i, v_j, v_k)$;
            end if
          end for
        end while
        if $|V(G^\Delta)| = n$ ; then
          return $(G^\Delta, G_2, S)$ ;
        end if
      end if
    end for
  end for
```

the pair $(\overline{P}, G^{\Delta})$ represents a *stably persistent formation*. The rest of this work describes how to achieve our formation goals with these formations.

CHAPTER 5

CONTROL LAWS FOR MULTI-ROBOT NETWORK FORMATIONS WITH PERSISTENT NETWORK GRAPHS

In Chapter 4, we presented an algorithm for determining if a desired formation is stably, persistently feasible given the proximity range of the multi-robot network, as well as efficiently, automatically generating a stable, persistent formation graph that respect the network's proximity range. Since these graphs are acyclic, we can automatically define control laws for their implementation while avoiding instabilities that are difficult to predict [15, 16]. These graphs can also be assembled entirely from vertex additions, as presented in Chapter 4.4.

In this chapter, we present control laws for multi-robot networks with stable, persistent network graphs. In Chapter 5.1, we review some relevant properties of stable, persistent graphs. We review the definition of a network deployment in Chapter 5.2. How robots use their local geometry and the relative positions of other robots to estimate their deployment is discussed in Chapter 5.3. Chapter 5.4 defines control laws for multi-robot networks with stable, persistent network graphs. Simulation results are presented in Chapter 5.5.

5.1 Stable, Persistent Graphs

In this chapter, we assume that the network already has a stable, persistent network graph G, and that each robot is aware of its local topology. Since the network graph G is a stable, persistent graph, it has several useful properties. As mentioned in Chapter 4.4, G is a Directed Acyclic Graph (DAG), and can be assembled entirely from vertex additions from a leader-first-follower seed graph. In this chapter, we assume that the leader robot is robot 1, and the first-follower robot is robot 2. Therefore, robot 1 has no constraints, robot 2 has one constraint such that $(v_2, v_1) \in E(G)$.

For all robots k such that $k \in N\backslash\{1, 2\}$, there exists $(v_i, v_j) \in V(G)\times V(G)$ such that $v_i \neq v_j$

and $\{(v_k, v_i), (v_k, v_j)\} \subset E(G)$. This implies that robots i and j are immediate successors of robot k. There is also a directed path from every vertex in $V(G)$ to the leader and first-follower [23]. Thus, all such followers k are predecessors of the leader and first-follower in the network. This implies that we can order the indices to correspond to a topological order of the network graph such that, $\forall (v_j, v_i) \in E(G)$, $i < j$. In this chapter, in order to facilitate future developments, we assume that the network is indexed in this manner.

5.2 Network Deployment

Recall from Chapter 2.2 that a network deployment of n robots is defined by a set of n *deployment positions* such that, $\forall i \in N$, $p_i : T \mapsto \mathbb{R}^2$ represent the desired trajectory for the robot assigned position i. These deployment positions define the *network deployment* $P : T \mapsto \mathbb{R}^2$ such that, $\forall t \in T$, $P(t) = \left[p_1(t)^T, \ldots, p_n(t)^T\right]^T$. For the deployment to be valid, $\forall (i, j, t) \in N \times N \times t$, $\|p_i(t) - \bar{p}_j(t)\| = \|\bar{p}_i(0) - \bar{p}_j(0)\|$ (i.e., all inter-position distances are preserved for the entire trajectory). In this manner, P captures both the desired formation ($\bar{P} = P(0)$) and the desired location of the formation in the environment as a function of time.

We assume that the leader and first-follower robots have the network deployment available to them, and that they can estimate their relative position to the network deployment. This implies that we can define the leader's control u_1 as a function of $(x_1 - p_1)$ and the first-follower's control u_2 as a function of $(x_2 - p_2)$. In Chapter 5.4, for all other robots, we define their control laws as a function of the desired formation $\bar{P}$ and its constraints in the network and the relative position of the corresponding robots. Since this is a persistent formation then, $\forall k \in N \setminus \{1, 2\}$, $\exists (i, j) \in N \times N$ such that $i \neq j$ and $\{(v_k, v_i), (v_k, v_j)\} \subset E(G)$, and we define $\dot{x}_k = u_k$ as a function of x_i and x_j.

These assumptions imply that the control laws we present here are applicable in a variety of situations. For example, if the NASA geologists explicitly define a deployment as the positions in the environment that the robots must satisfy, then these assumptions and control laws imply that only the leader and first-follower must be aware of the deployment P, and that only the leader and first-follower require localization ability (i.e., the ability to estimate their position relative to the environment). For the rest of the network, each robot successfully navigates with only knowledge of the desired local geometry defined in $\bar{P}$ and the relative positions of its immediate successors in the network. Therefore, if the leader and first-follower coordinate and decide to deviate from the plan, the rest of the formation will follow.

The latter situation (where the leader and first-follower decide to deviate from the plan) corresponds to a special case where the leader-first-follower pair of robots decide where the robots should position the formation. For example, assume that the leader robot determines a deployment for the network and defines it based on its own location. IN this case, $x_1 = p_1$, and the leader coordinates with the first-follower to define p_2 as a function of p_1. Thus, our approach to defining these control laws is applicable to many situations, without requiring localization, and with little or no planning required.

5.3 Local Geometry and Circle-Circle Intersection Solutions

Here, we describe how each follower robot estimates its desired location and dynamics in the formation. This involves determining a velocity that satisfies the deployment using the desired formation $\bar{P}$, as well as the positions and velocities of its adjacent neighbors in the network graph.

In this chapter, we assume that the deployment positions are indexed in the same manner as the robots assigned to them such that, $\forall i \in N$, robot i is assigned position i. The desired formation $\bar{P}$ and the network graph G define *desired edge weights* for all $(v_i, v_j) \in E(G)$. Thus, we define a *desired length function* $d : V(G) \times V(G) \times \mathbb{R}^{2n} \mapsto \mathbb{R}^{+}$ such that, $\forall (v_i, v_j) \in V(G) \times V(G)$, $d(v_i, v_j, \bar{P}) = \|\bar{p}_i - \bar{p}_j\|$. For each pair $(v_i, v_j) \in V(G) \times V(G)$, we define $d_{ij} = d(v_i, v_j, \bar{P}) = \|\bar{p}_i - \bar{p}_j\|$ for simpler notation. Hence, d assigns a desired weight to each edge in the network graph.

Recall from Chapter 2.4.1 that the network graph G and the network trajectory $X(t)$ define a weight function such that, $\forall (v_i, v_j) \in V(G) \times V(G)$, $\delta_{ij} = \delta(v_i, v_j, X(t)) = \|x_i(t) - x_j(t)\|$. The network graph is persistent such that, when initially in formation, the network *maintains the formation* if, $\forall (v_i, v_j) \in G$, $\delta_{ij} = \|x_i(t) - x_j(t)\| = d_{ij}$ $\forall t \in T$. However, there may be some initial formation error, implying that simply maintaining the initial inter-robot distances corresponding to the network graph may not satisfy the desired formation. Further, if error is introduced into the formation, we would not desire for the control laws to attempt to maintain the error. We must control each robot to satisfy its unique, local geometry with its adjacent neighbors in a manner such that the global network geometry satisfies the formation and the deployment.

The leader and first-follower robots attempt to satisfy this by stabilizing to their deployment positions. As stated in Chapter 5.2, we assume that the leader robot 1 and first-follower robot 2 have direct access to their deployment positions p_1 and p_2. Recall that, since each additional robot (i.e., each follower robot) is added by a vertex addition, then, $\forall k \in N \setminus \{1, 2\}$, there exists $(v_i, v_j) \in V(G) \times V(G)$ such that $v_i \neq v_j$ and $\{(v_k, v_i), (v_k, v_j)\} \subset E(G)$. In this case, robots i and j are immediate successors of robot k.

To determine how to satisfy their local geometry, each follower robot employs solutions to the circle-circle intersection problem to estimate their deployment positions and their dynamics, shown in Figure 5.1. We define

$$Q_1 = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}, \qquad Q_2 = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix},$$

and choose $Q_k \in \mathbb{R}^{2\times 2}$ such that $Q_k = Q_1$ or Q_2. Given this choice of Q_k, two circles whose centers are at $x_i(t) \in \mathbb{R}^2$ and $x_j(t) \in \mathbb{R}^2$ with radii of d_{ik} and d_{jk} respectively intersect at the

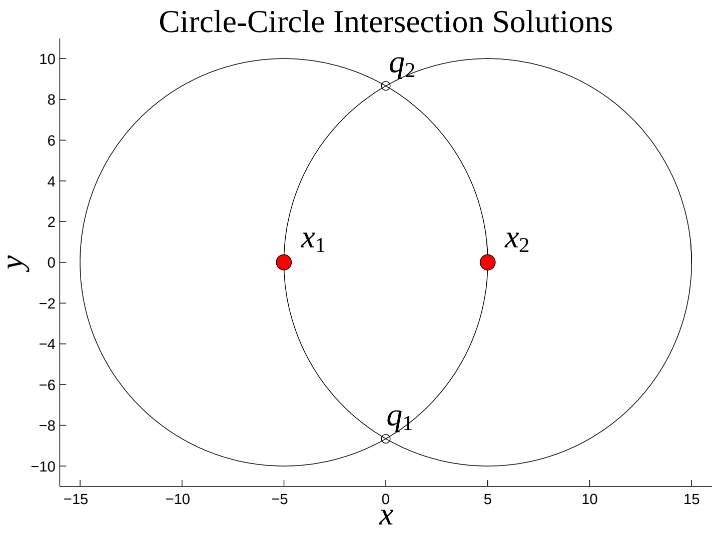


Figure 5.1. The circle-circle intersection solutions for robot 3 with $x_1 = [-5, 0]^T$, $x_2 = [5, 0]^T$, and $d_{31} = d_{32} = 10$. Here, $q_1 = f_k(x_1, x_2, d_{31}, d_{32}, Q_1)$, and $q_2 = f_k(x_1, x_2, d_{31}, d_{32}, Q_2)$. Since the circle-circle intersection problem typically has two solutions defined by two equations, it is necessary to define f_k to correspond to the correct equation for each robot k and its geometry in the formation.

point defined by

$$f_k(x_i(t), x_j(t), d_{ik}, d_{jk}, Q_k) = \frac{x_i(t) + x_j(t)}{2} + \frac{1}{2\delta_{ij}^2}\left(\left(d_{ik}^2 - d_{jk}^2\right)\left(x_j(t) - x_i(t)\right) + Q_k\left(x_j(t) - x_i(t)\right)\sqrt{\left(\left(d_{ik} + d_{jk}\right)^2 - \delta_{ij}^2\right)\left(\delta_{ij}^2 - \left(d_{jk} - d_{ik}\right)^2\right)}\right). \tag{6}$$

The choice of Q_k selects which intersection point is desired, since there are (typically) two solutions to the circle-circle intersection problem. For each follower robot k, the appropriate equation is used to define f_k. Since f_k is defined to choose the appropriate circle-circle intersection point, then, $\forall\{(i, j, k) \in N \times N \times N : i \neq j \neq k\}$, $\bar{p}_k = f_k(\bar{p}_i, \bar{p}_j, d_{ij}, d_{ik}, Q_k)$. Also, if $\|x_i(t) - x_j(t)\| = \|\bar{p}_i - \bar{p}_j\|$, then $x_k(t) = f_k(x_i(t), x_j(t), d_{ik}, d_{jk}, Q_k)$ satisfies the same local geometry with $x_i(t)$ and $x_j(t)$ as does $\bar{p}_k$ with $\bar{p}_i$ and $\bar{p}_j$. Thus, the robot assigned position k can determine its desired position without using any localization ability. Each follower robot can determine its desired position based only on the estimated range and bearing of its adjacent neighbors in the network graph. In this work, we say $x_k = f_k(x_i, x_j)$ for simpler notation to define these circle-circle intersection solutions.

Note that, when operating on $x_i(t)$ and $x_j(t)$, there is a discontinuity when $x_i(t) = x_j(t)$

(i.e., when the circles have a common center). There are also complex solutions when

$$\left\| d_{ki} + d_{kj} \right\|^2 < \left\| x_j(t) - x_i(t) \right\|^2$$

or

$$\left\| d_{kj} - d_{ki} \right\|^2 > \left\| x_j(t) - x_i(t) \right\|^2 .$$

The first condition indicates that the centers of the circles are too far apart to intersect. The second condition implies that one circle is inside the other so that they do not intersect.

5.4 Multi-Robot Network Control Laws with Stable, Persistent Network Graphs

Here, we define control laws for multi-robot networks with stable, persistent network graphs. First, we define the formation error. Then, we present control laws for two cases. In the first case, we assume that each robot has access to its deployment position. While this involves more knowledge for the follower robots than we previously assume, it is useful for setting up the second case, where only the leader and first-follower have knowledge of their deployment positions.

5.4.1 The Formation Error

Since the deployment P represents the desired network trajectory, we can define the relative error of each specific robot as $\tilde{x}_i = x_i - p_i \ \forall i \in N$. The *network error* $\tilde{X} : T \mapsto \mathbb{R}^{2n}$ is defined such that $\tilde{X}(t) = \left[\tilde{x}_1(t)^T, \ldots, \tilde{x}_n(t)^T\right]^T$. These error definitions are useful for characterizing how the formation error behaves given the control laws of the network.

We always assume that the leader and first-follower have access to their deployments. We define persistent control strategies for two cases. In the first case, we assume that $\forall k \in N$ such that distinct robots i and j are immediate successors of robot k, follower robot k has access to p_i and p_j, and can thus derive $p_k = f_k(p_i, p_j)$. Here, the robots are controlled with knowledge of their specific deployment positions and their dynamics. In the second case, we assume that each follower robot k can only estimate p_k by the actual positions of the other robots as $\hat{p}_k = f_k(x_i, x_j)$. Here, the robots are controlled by estimating their deployments and dynamics using sensor data.

5.4.2 Control With Knowledge of the Deployment Positions and Their Dynamics

We assume that the leader robot 1 and the first-follower robot 2 have access to p_1 and p_2 and can share them with neighboring robots. For our first scenario, we assume that, if $n \geq 3$, robots 1 and 2 communicate $(p_1, \dot{p}_1, p_2, \dot{p}_2)$ to robot 3. This implies that robot 3 can calculate both p_3 and $\dot{p}_3$, and can share those with its adjacent neighbors. In general, we assume that, $\forall k \in N \setminus \{1, 2\}$ such that distinct robots i and j are immediate successors of robot k, robots i and j share $(p_i,\ \dot{p}_i,\ p_j,\ \dot{p}_j)$ with robot k. This could be accomplished by robot sharing these as explicit functions of time, or communicating these values as the formation is executed. Then, robot k can derive $p_k = f_k(p_i, p_j)$ and $\dot{p}_k = \frac{\partial f_k}{\partial p_i}\left(p_i, p_j\right)\dot{p}_i + \frac{\partial f_k}{\partial p_j}\left(p_i, p_j\right)\dot{p}_j$. We can define each robots's control based on its desired state and the desired state's dynamics.

For all $i \in N$, we define robot i's control by $u_i = \dot{p}_i - K_i p_i$, where $-K_i \in \mathbb{R}^{2\times 2}$ is a Hurwitz matrix. This implies that the error dynamics are

$$\dot{\tilde{x}}_i = \dot{x}_i - \dot{p}_i = u_i - \dot{p}_i = \dot{p}_i - K_i\tilde{x}_i - \dot{p}_i = -K_i\tilde{x}_i.$$

The system $\dot{\tilde{x}}_i = -K_i\tilde{x}_i$ has a globally exponentially stable origin. Further, the network error is described by the system $\dot{\tilde{X}} = -K\tilde{X}$, where $-K \in \mathbb{R}^{2n\times 2n}$ is a Hurwitz matrix with $-K_1, \ldots, K_n$ on its diagonal. Therefore, the network error has a globally exponentially stable origin. Hence, $\lim_{t\to\infty} \tilde{X}(t) = 0$.

5.4.3 Control Without Knowledge of the Deployment Positions and Their Dynamics

In the previous case, we assumed that the leader and first-follower robots where sharing their deployment positions p_1 and p_2 with their adjacent neighbors. Without localization ability, robots sharing this information must come to an agreement about their locations in a common frame of reference before such communicated locations and velocities are meaningful. This suggests that either all robots have localization ability, or that each edge in the network graph implies a high level of coordination between those pairs of robots. The in-degree of stable, persistent network graphs is only bounded by $n - 1$. Since the robots have limited computation and communication abilities, this could be problematic for a robot who is the immediate successor of potentially $n - 1$ robots in the network.

As an alternative, we now consider the case of estimated knowledge of the deployment P. We assume that the leader robot 1 has access to p_1, and the first-follower robot 2 has access to p_2. However, follower robots k with immediate successors i and j do not have access to either p_i, p_j, or p_k.

For all robots k such that robots i and j are its distinct, immediate successors, we assume that robot k has access to the local geometry that must be satisfied (i.e., $\overline{p}_i$, $\overline{p}_j$, and $\overline{p}_k$), as well as x_i and x_j, as in the case where robot k estimates the relative positions of robots i and j through sensor information. For all robots $i \in N$, we define its *estimated deployment* $\hat{p}_i : T \mapsto \mathbb{R}^2$. For all follower robots k with distinct, immediate successors i and j, robot k estimates its deployment position by $\hat{p}_k = f_k\left(x_i, x_j\right)$ and $\dot{\hat{p}}_k = \frac{\partial f_k}{\partial x_i}\left(x_i, x_j\right)\dot{x}_i + \frac{\partial f_k}{\partial x_j}\left(x_i, x_j\right)\dot{x}_j$. Since the leader and first-follower robots still have access to p_1 and p_2, then $\hat{p}_1 = p_1$ and $\hat{p}_2 = p_2$. We also define an estimation of the desired deployment as $\hat{P} : T \mapsto \mathbb{R}^{2n}$ such that $\hat{P}(t) = [\hat{p}_1(t)^T, \ldots, \hat{p}_n(t)^T]^T$. Similarly, $\forall i \in N$, we define an *estimation of the error* of robot i as $\hat{x}_i = x_i - \hat{p}_i$, and the *estimated network error* as $\hat{X} = X - \hat{P}$.

For all $i \in N$, we define robot i's control law by $u_i = \dot{\hat{p}}_i - K_i\hat{x}_i$, where $-K_i \in \mathbb{R}^{2\times 2}$ is a Hurwitz matrix. This implies that $\dot{\hat{x}}_i = -K_i\dot{\hat{x}}_i \; \forall i \in N$, which has a globally stable origin. This implies that the *estimated* network error $\hat{X}$ has a globally, exponentially stable origin and $\lim_{t\to\infty} \hat{X}(t) = 0$.

If, $\forall k \in N$ such that $\{(v_k, v_i), (v_k, v_j)\} \subset E$, we assume that $\lim_{t\to\infty} \tilde{x}_i(t) = \lim_{t\to\infty} \tilde{x}_j(t) =$

0, then this implies that

$$\begin{aligned}\lim_{t\to\infty}\left(\hat{p}_k(t)-p_k(t)\right) &= \lim_{t\to\infty}\left(f_k\left(x_i(t),x_j(t)\right)-f_k\left(p_i(t),p_j(t)\right)\right)\\ &= \lim_{t\to\infty}\left(f_k\left(p_i(t),p_j(t)\right)-f_k\left(p_i(t),p_j(t)\right)\right)\\ &= 0.\end{aligned}$$

Also,

$$\begin{aligned}\lim_{t\to\infty}\tilde{x}_k(t) &= \lim_{t\to\infty}\left(x_k(t)-p_k(t)\right)\\ &= \lim_{t\to\infty}\left(x_k(t)-\hat{p}_k(t)+\hat{p}_k(t)-p_k(t)\right)\\ &= \lim_{t\to\infty}\left(\hat{x}_k(t)+\hat{p}_k(t)-p_k(t)\right)=0.\end{aligned}$$

Note that, by our assumptions, $\hat{p}_1 = p_1$ and $\hat{p}_2 = p_2$. This implies that $\lim_{t\to\infty}\tilde{x}_1(t) = \lim_{t\to\infty}\tilde{x}_2(t) = 0$. Recall that the network graph is a directed, acyclic graph such that the leader and first-follower are successors to every other robot in the network. Then, by the topological properties of the network graph G and induction, this implies that, $\forall k \in N$ such that $k \geq 3$, $\exists (i, j) \in N \times N$ such that $i \neq j$, $\{(v_k, v_i), (v_k, v_j)\} \subset E$, $i < k$, $j < k$, and $\lim_{t\to\infty}\tilde{x}_i(t) = \lim_{t\to\infty}\tilde{x}_j(t) = 0$. Hence, $\lim_{t\to\infty}\tilde{X}(t) = 0$.

However, the equations for f_k are nonlinear. As discussed in Chapter 5.3, there can be discontinuities or imaginary components of the solution in the presence of specific errors. Further, since δ_{ij} appears in the denominator of (6), this implies that the value may approach infinity as δ_{ij} approaches zero. Therefore, we have only *local stability* provided by these control laws. In order to be effective, we must first *assemble* these formations to with a sufficiently small initial error before we employ these control laws such that we operate within a sufficiently tight neighborhood of P. This is part of our motivation for a strategy for formation assembly, presented in Chapter 6.

5.5 Deployment Simulations

Here, we present simulation results for the control strategy using estimations of the desired states presented in Chapter 5.4.3. First, we present a scenario of the NASA project. Using a Graphical User Interface (GUI) for entering formation positions, we assume that the geologists specify a hexagonal formation of $n = 7$ robots to track a moving terrain feature with their sensors. A minimally persistent network graph G is generated such that, for each edge $(v_i, v_j) \in E(G)$, $d_{ij} = 10$ m. The software automatically configures the robots with the control laws in Chapter 5.4.3 and the network graph G. The robots then implementing the control laws begin collecting data. Initially, during the data collection, the terrain feature begins moving with a velocity of $[1, 1]^T$. The leader and first-follower begin moving in order to track this motion, defining a desired velocity of $\dot{p}_i = [1, 1]^T$, $\forall i \in N$. For each robot i, we define $-K_i = -I$ and implement the control laws in Chapter 5.4.3.

The simulation results for this motion are shown for three different initial error assumptions. First, it is assumed that the initial error of each robot is bounded to within 2 m of $p_i(0)$ $\forall i \in N$. Then, we perform two simulations to test the "robustness" of the control laws when the conditions discussed in Chapter 5.3 occur. In one simulation, it is assumed that all the robots are initially very close to the initial desired state of the leader such that, $\forall (i, j) \in N \times N$, $x_i(0)$, $x_j(0)$ are within 10^{-60} m of each other. In the last simulation, we assume that the initial position of each robot is within 20 m of the initial desired leader state, but allow large initial formation errors. These last two scenarios allow sufficient error to produce complex results for the circle-circle intersection solutions.

5.5.1 Simulation Results

Figure 5.2 shows the resulting network trajectory when the error of each robot is initially bounded to 2 m. In this figure, dashed lines represent the desired trajectory, while solid lines represent the actual trajectory. Arrows between the states correspond to the edges of the network graph G.

Figure 5.3 shows the errors of each robot in the network. Figure 5.3(a) shows the error of each robot with respect to their desired states, which approach zero as $t \to \infty$. Figure 5.3(b) shows the errors of each constraint of the network, which also approach zero as $t \to \infty$.

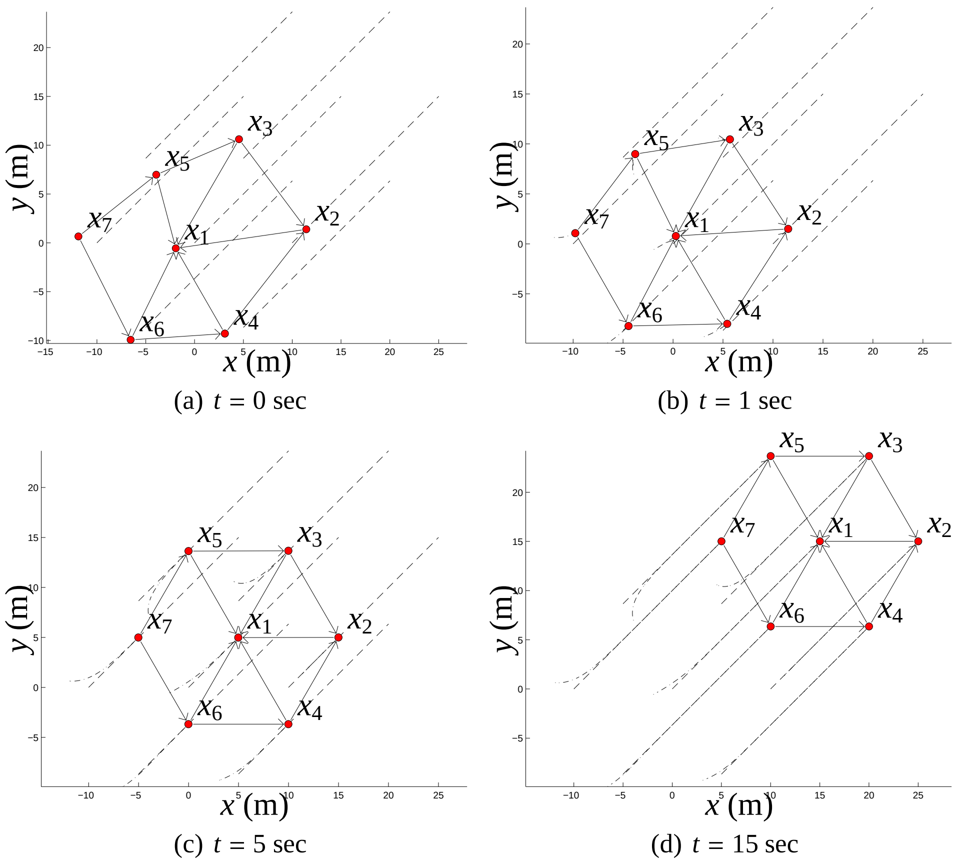

(a) $t = 0$ sec (b) $t = 1$ sec (c) $t = 5$ sec (d) $t = 15$ sec

Figure 5.2. The network trajectory. In this figure, dashed lines represent the desired deployment, while solid lines represent the actual trajectory. Arrows between the states correspond to the edges of the network graph. Here, we define an initial error of 2 m for each robot. As the robots move, each robot stabilizes to its deployment.

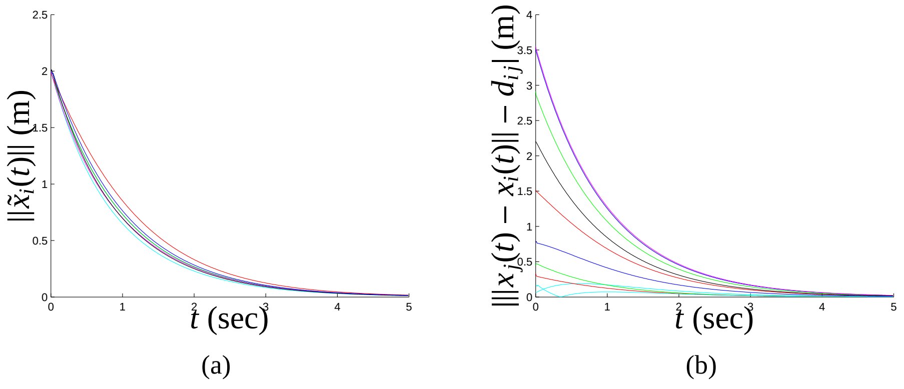

(a) (b)

Figure 5.3. The network error and constraint errors. Here, we define an initial error bound of 2 m for each robot. As the robots move, the error in the formation approaches zero as $t \rightarrow \infty$. Figure 5.3(a) shows the formation errors of each robot, and Figure 5.3(b) shows the errors of each robot in satisfying their constraints. All errors stabilize to zero as $t \rightarrow \infty$.

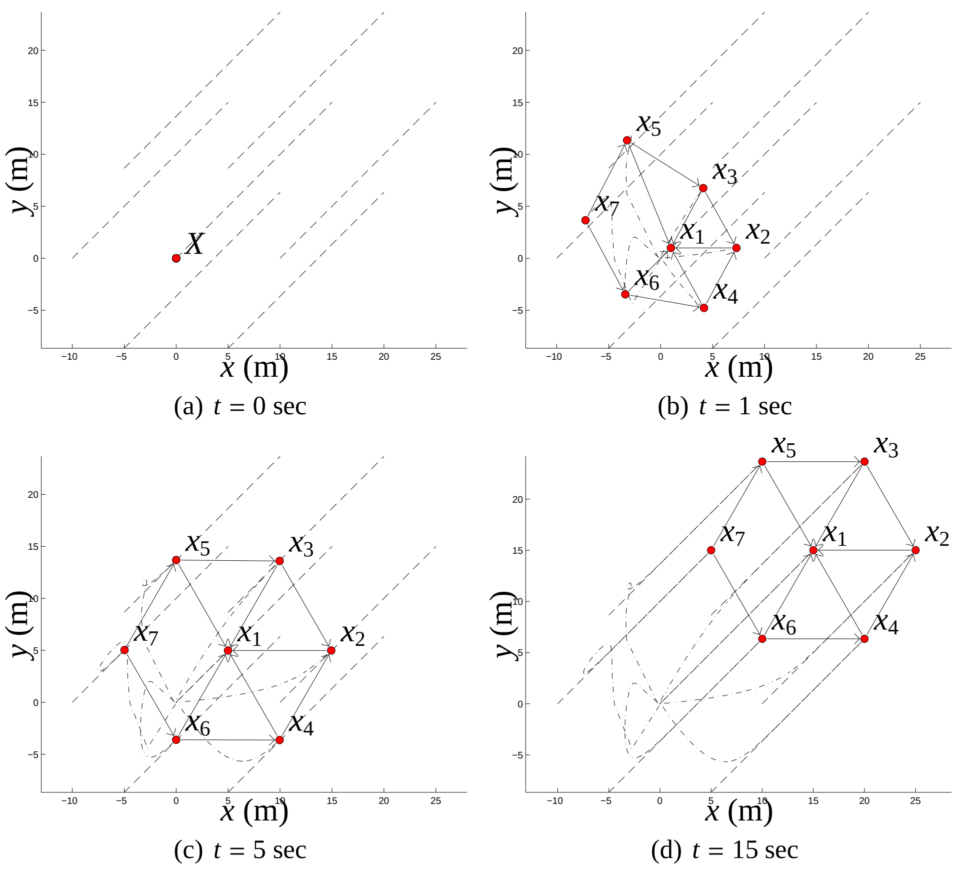

(a) $t = 0$ sec (b) $t = 1$ sec (c) $t = 5$ sec (d) $t = 15$ sec

Figure 5.4. The network trajectory. In this figure, dashed lines represent the desired deployment, while solid lines represent the actual trajectory. Arrows between the states correspond to the edges of the network graph. Here, each robots initial state at time $t = 0$ to be within 10^{-60} m of $p_1(0)$. As the robots move, each robot stabilizes to its deployment.

In the next simulation, we force each robot's initial state at time $t = 0$ to be within 10^{-60} m of $p_1(0)$. This can produce complex results, large errors in the estimation of the network error, and, thus, large errors in the early portions of the formation trajectory. However, the errors still stabilize to zero as $t \to \infty$. Figure 5.4 shows the corresponding network trajectory of this scenario. The robots still stabilize to their desired deployment.

Figure 5.5 shows the corresponding errors for each robot, as well as each constraint in the network. The errors still approach 0 as $t \to \infty$.

To further test the robustness of the dynamics defined by f_k when it produces complex results, we enlarge bound on the initial error of each robot to 20 m. Figure 5.6 shows the network trajectory for this scenario. As $t \to \infty$, the robots still stabilize to their desired

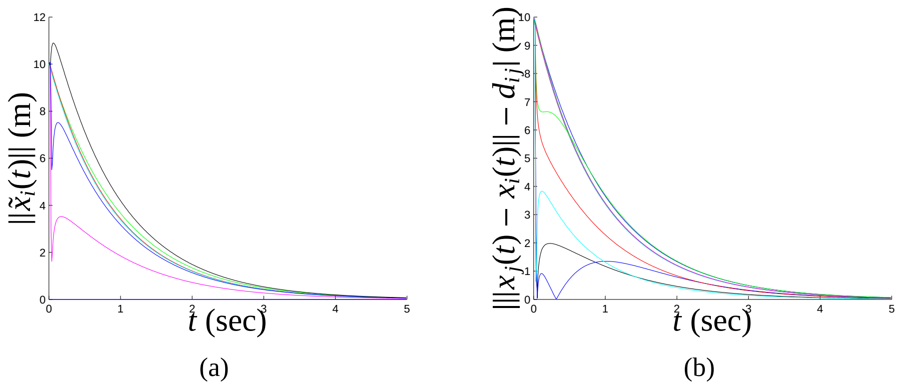
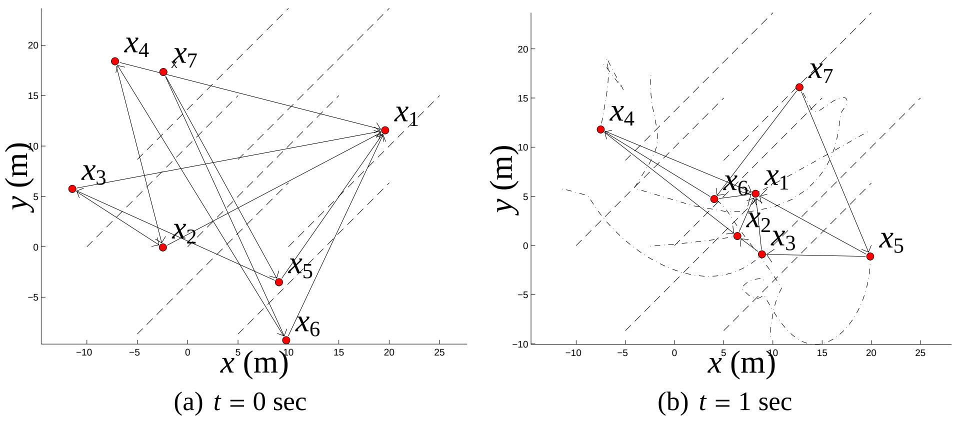


Figure 5.5. The network error and constraint errors. Here, each robots initial state at time $t = 0$ to be within 10^{-60} m of $p_1(0)$. As the robots move, the error in the formation approaches zero as $t \to \infty$. Figure 5.5(a) shows the formation errors of each robot, and Figure 5.5(b) shows the errors of each robot in satisfying their constraints. All errors stabilize to zero as $t \to \infty$.

deployments.

As seen in Figure 5.7, this produces large initial errors, as well as complex results. However, using these dynamics, the errors still stabilize to zero as $t \to \infty$.

These simulation results shows us that we need an alternative method to assemble a persistent formation before these control laws are implemented. This is the subject of the next chapter.



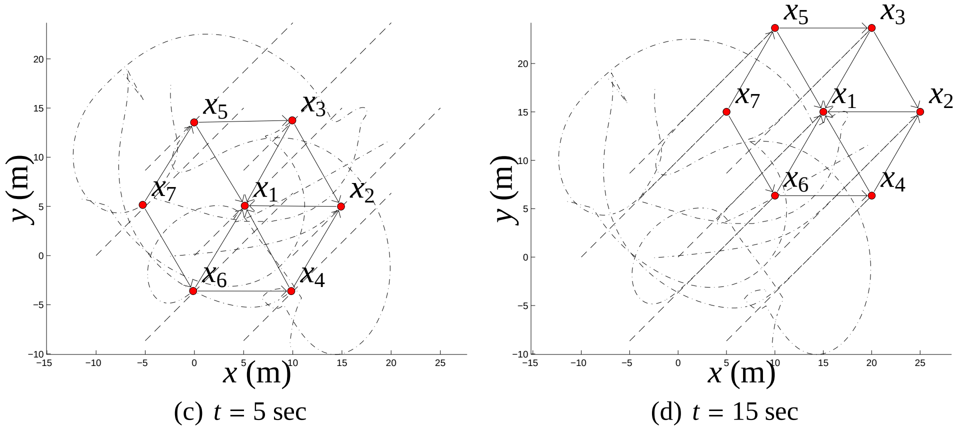


Figure 5.6. The network trajectory. In this figure, dashed lines represent the desired deployment, while solid lines represent the actual trajectory. Arrows between the states correspond to the edges of the network graph. Here, we bound the initial error of each robot to 20 m. As the robots move, each robot stabilizes to its deployment.

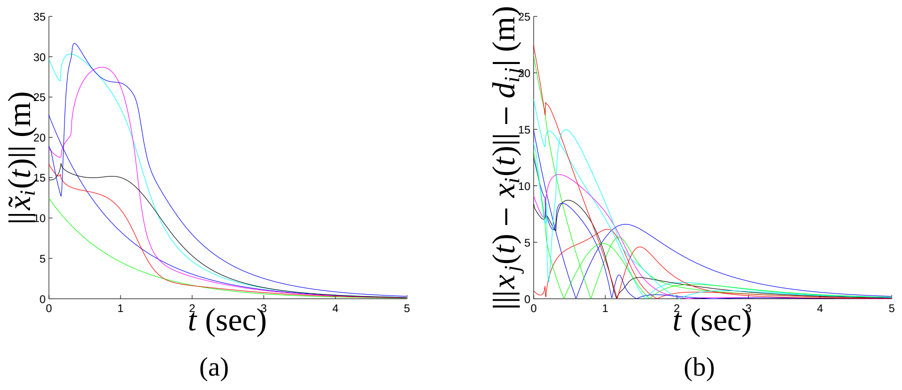


Figure 5.7. The network error and constraint errors. Here, each robots initial error is bounded to within 20 m. As the robots move, the error in the formation approaches zero as $t \to \infty$. Figure 5.7(a) shows the formation errors of each robot, and Figure 5.7(b) shows the errors of each robot in satisfying their constraints. All errors stabilize to zero as $t \to \infty$.

CHAPTER 6

FORMATION ASSEMBLY WITH EMBEDDED GRAPH GRAMMAR SYSTEMS (EGGS)

In Chapter 5, we present control laws for deploying multi-robot networks with stably persistent formation graphs. While these control laws featured unique equilibria for each robot in the formation, they were only locally stable. As such, they should only be used in deploying formations with sufficiently small initial error. In this chapter, we present a way of *assembling formations* such that we can bound the initial formation error before deployment.

As a starting point, we assume that the NASA scientists enter a stably, persistently feasible desired formation as a set of points in a Graphical User Interface (GUI), given the network's proximity range. We assume this GUI program implements the methods from Chapter 4, generating a stable, persistent formation graph G^{Δ}, along with a leader-first-follower seed graph G_2 and a sequence of vertex addition operations for assembling G^{Δ} from G_2. Figure 6.1 shows an example of this GUI, with desired formation $\bar{P}$ and corresponding stable, persistent network graph G^{Δ}.

In this chapter, we present a method for automatically generating Embedded Graph Grammar (EGG) systems for assembling formations using these vertex operation sequences. This involves defining EGG rules and control laws for coordinating leader-first-follower pairs and performing vertex additions.

In Chapter 6.1, we present control laws for implementing leader-first-follower pairs, as well as performing vertex additions. Chapter 6.2 shows how to define an EGG system given a stably, persistently feasible desired formation $\bar{P}$, a leader-first-follower seed graph G_2, and a sequence of vertex addition operations that assemble a stable, formation graph G^{Δ} from G_2. This EGG assigns each robot to a position in the formation and provides control

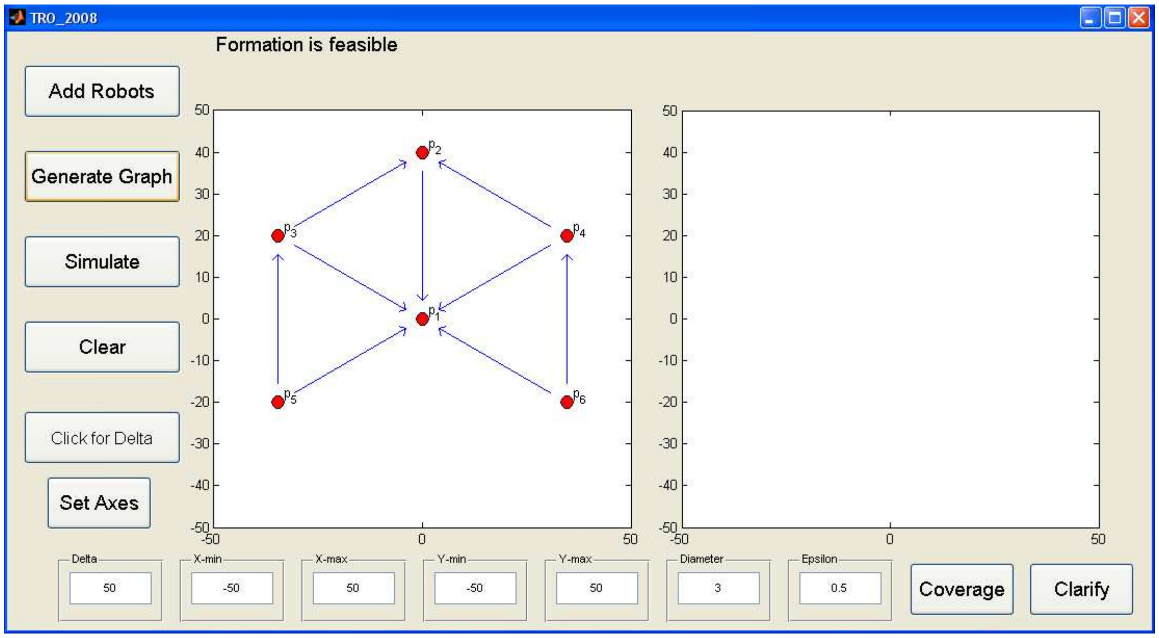


Figure 6.1. The Graphical User Interface (GUI). This program implements the methods from Chapter 4 to generate a stable, persistent network graph G^{Δ}.

laws that allow them to achieve the correct relative geometry specified in the desired formation. A method for allowing unassigned robots to navigate through the network to locations where they can apply vertex additions is presented in Chapter 6.3. In Chapter 6.4, we show that we can arbitrarily bound the final error in the formation assembly. Chapter 6.5 presents our methods for implementing the resulting EGG system on decentralized networks, where decision-making is distributed across the network. This involves the presentation of decentralized EGG rule evaluation. Simulations results for a large scale network are described in Chapter 6.6. Two scenarios with a prototype multi-robot network are presented in Chapter 6.7.

6.1 Control Laws For Assembling Persistent Formations

Here, we present control laws for building the leader-first-follower seed graph and performing vertex addition operations. We define control laws for assembling these formations that satisfy the proximity range constraints and the maximum velocity constraints, and prove their stability.

Recall from Chapter 4.4 that, given the desired formation and the network's proximity range, the $StablePersistentDelta$ returns a stable, minimally persistent formation graph G^{Δ}, along with a leader-first-follower seed graph G_2 and a sequence of vertex addition operations. These vertex addition operations, when performed on G_2, result in a sequence of graphs $(G_2, \ldots, G_n)$ such that $G_n = G^{\Delta}$, the desired stable, minimally persistent formation graph. The pair $(\bar{P}, G^{\Delta})$ is a *persistent formation*, with a desired geometry specified by $\bar{P}$ and a desired network graph specified by G^{Δ}.

First, we define control laws for a leader and first-follower pair of robots to establish the leader-first-follower seed graph. Then, we present control laws for robots performing vertex addition operations.

6.1.1 Leader-First-Follower Control Laws

In order to assemble the leader-first-follower seed graph, we must navigate a first-follower robot such that it satisfies the distance defined in $\bar{P}$ with the leader robot. The following theorem presents control laws that accomplish this while respecting the limitations of the network.

Theorem 6.1 *For a multi-robot network with proximity range $\Delta \in \mathbb{R}^+$, consider a stable, persistent formation $(\bar{P}, G^{\Delta})$ with leader and first-follower positions $\bar{p}_a$ and $\bar{p}_b$, respectively. Assume that robot a is assigned position $\bar{p}_a$, and that robot b is assigned position $\bar{p}_b$ in desired formation $\bar{P}$. For a proximity range $\Delta \in \mathbb{R}^+$ and a maximum velocity $u_{max} \in \mathbb{R}^+$, robot b's control laws are defined by*

$$d_{ab} = ||\bar{p}_a - \bar{p}_b||, \tag{7}$$

$$p_b = d_{ab} \frac{x_b - x_a}{||x_b - x_a||} + x_a, \tag{8}$$

$$K = \frac{u_{max}}{\Delta}, \tag{9}$$

$$u_b = -K(x_b - p_b) = -K\left(1 - \frac{d_{ab}}{||x_b - x_a||}\right)(x_b - x_a). \tag{10}$$

We assume that $u_a(t) = 0 \ \forall t \in T$. Then, for every initialization of the pair such that

$0 < \|x_b(0) - x_a\| \leq \Delta$,

- x_b *is continuously differentiable,*
- $\dot{p}_b(t) = 0\ \forall t \in T$*, and* p_b *is a globally, exponentially stable equilibrium point of* x_b*,*
- $\lim_{t\to\infty} \|x_b(t) - x_a\| = d_{ab}$*,*
- $\|x_b(t) - x_a\| \leq \Delta\ \forall t \in T$*,*
- $\|u_b(t)\| \leq u_{max}\ \forall t \in T$*, and*
- x_b *is Lipschitz continuous.*

Proof: First, we show that $x_b(t) \neq x_a\ \forall t \in T$, and x_b is continuously differentiable. By our assumptions, $x_a(t)$ is a constant $\forall t \in T$. Also from our assumptions, $x_b(0) \neq x_a$, and $d_{ab} > 0$. By (8) and (10), x_b is continuously differentiable if we can guarantee that $x_b(t) \neq x_a\ \forall t \in T$. To see the behavior of $\|x_b - x_a\|$, note that

$$\frac{d}{dt}\left(\frac{1}{2}\|x_b - x_a\|^2\right) = (x_b - x_a)^T(\dot{x}_b - \dot{x}_a) = (x_b - x_a)^T u_b.$$

Substituting (10) into the previous yields

$$\begin{aligned}\frac{d}{dt}\left(\frac{1}{2}\|x_b - x_a\|^2\right) &= -K\left(1 - \frac{d_{ab}}{\|x_b - x_a\|}\right)(x_b - x_a)^T(x_b - x_a)\\ &= -K\left(1 - \frac{d_{ab}}{\|x_b - x_a\|}\right)\|x_b - x_a\|^2.\end{aligned}$$

This implies that

$$\frac{d}{dt}\left(\frac{1}{2}\|x_b(t) - x_a\|^2\right) > 0 \text{ if } \|x_b(t) - x_a\| < d_{ab}. \tag{11}$$

Equation (11) implies that the distance between robots a and b is always increasing if their distance is less than d_{ab}. Thus, if we assume that, for some $t \in T$, that $\|x_b(t) - x_a\| = 0$, we always have a contradiction. If $t = 0$, then this violates our initial assumption that $\|x_b(0) - x_a\| \neq 0$. If $t > 0$, then there exists some $t_1 > 0$ such that $t_1 < t$, $\frac{d}{dt}\left(\frac{1}{2}\|x_b(t_1) - x_a\|^2\right) < 0$, and $\|x_b(t_1) - x_a\| < d_{ab}$. In other words, this implies that there exists a time before t where the distance between the robots is decreasing and their distance is less than d_{ab}. However, this violates (11). Therefore, $x_b(t) \neq x_a \; \forall t \in T$, and x_b is continuously differentiable over T.

To see that $\dot{p}_b = 0$, from (8) and the quotient rule, we have that

$$\begin{aligned}
\dot{p}_b(t) &= d_{ab}\frac{d}{dt}\left(\frac{x_b(t) - x_a}{\|x_b(t) - x_a\|} + x_a\right) \\
&= d_{ab}\frac{(\dot{x}_b(t) - \dot{x}_a)\,\|x_b(t) - x_a\| - (x_b(t) - x_a)\,\frac{d}{dt}\left(\|x_b(t) - x_a\|\right)}{\|x_b(t) - x_a\|^2} \\
&= d_{ab}\frac{u_b(t)\|x_b(t) - x_a\| - (x_b(t) - x_a)\frac{(x_b(t)-x_a)^T}{\|x_b(t)-x_a\|}u_b(t)}{\|x_b(t) - x_a\|^2} \\
&= \frac{d_{ab}}{\|x_b(t) - x_a\|^2}\left(\|x_b(t) - x_a\| - \frac{(x_b(t) - x_a)(x_b(t) - x_a)^T}{\|x_b(t) - x_a\|}\right)u_b(t).
\end{aligned}$$

Substituting (10) into the previous gives

$$\dot{p}_b(t) = \frac{-d_{ab}K}{\|x_b(t) - x_a\|}\left(1 - \frac{d_{ab}}{\|x_b(t) - x_a\|}\right)((x_b(t) - x_a) - (x_b(t) - x_a)) = 0.$$

Hence, $\dot{p}_b(t) = 0 \; \forall t \in T$, and p_b is a constant.

To see that p_b is a globally, exponentially stable equilibrium point of x_b, consider the translated system

$$\tilde{x}_b = x_b - p_b.$$

By (10), this implies that

$$\dot{\tilde{x}}_b = \dot{x}_b - \dot{p}_b = -K(x_b - p_b) = -K\tilde{x}_b,$$

This implies that $\tilde{x}_b$ has a globally, exponentially stable origin, and that p_b is a globally, exponentially stable equilibrium point of x_b.

Note from (8) that $p_b(t) \in \mathbb{R}^2$ is a point d_{ab} away from x_a and in the direction of x_b from x_a. Therefore, we have from (8) that

$$\lim_{t\to\infty}\|x_b(t) - x_a\| = \lim_{t\to\infty}\|p_b - x_a\| = d_{ab}.$$

To see that $\|x_b(t) - x_a\| \leq \Delta \; \forall t \in T$, note that the control laws of x_b imply that

$$x_b(t) - p_b = e^{-Kt}\left((x_b(0) - p_b)\right). \tag{12}$$

Therefore, $x_b(t) = e^{-Kt}(x_b(0) - p_b) + p_b$, and the range of x_b is the "straight" line segment

$$X_b = \{x_b(0)s + (1-s)p_b : s \in (0,1]\}.$$

Let $B_\Delta[x_a]$ define the *closed ball* of radius Δ centered x_a. From our initial assumptions, $\|x_b(0) - x_a\| \leq \Delta$, and $x_b(0) \in B_\Delta[x_a]$. Since $\bar{P}$ describes a stably, persistently feasible formation for proximity range Δ, then $d_{ab} \leq \Delta$. From (8), this implies that $\|p_b - x_a\| \leq \Delta$, and $p_b \in B_\Delta[x_a]$. Since X_b is a line segment between $x_b(0)$ and p_b, and $\{p_b, x_b(0)\} \subset B_\Delta[x_a]$, then the convexity of $B_\Delta[x_a]$ implies that $X_b \subset B_\Delta[x_a]$. Thus, $\|x_b(t) - x_a\| \leq \Delta \; \forall t \in T$.

To prove that $\|u_b(t)\| \leq u_{max} \; \forall t \in T$, we define $p_\Delta \in \mathbb{R}^2$ such that

$$p_\Delta = \Delta \frac{x_b(0) - x_a}{\|x_b(0) - x_a\|} + x_a.$$

In other words, p_Δ is a point Δ distance away from x_a, in the direction of $x_b(0)$ from x_a. Let R_Δ be the "straight" line segment such that

$$R_\Delta = \{x_a s + (1-s)p_\Delta : s \in [0,1]\}.$$

Since $\|p_\Delta - x_a\| = \Delta$, then R_Δ describes a line of length Δ. Since $p_b(0)$ is a point d_{ab} away from x_a in the direction of $x_b(0)$ from x_a, then $p_b(0) \in R_\Delta$. Since $\|x_b(0) - x_a\| \leq \Delta$, then $x_b(0) \in R_\Delta$. Since R_Δ describes a line of length Δ and $\{x_b(0), p_b(0)\} \subset R_\Delta$, then $\|x_b(0) - p_b(0)\| \leq \Delta$.

Equation (12) implies that, $\forall t \in T$,

$$\|x_b(t) - p_b\| \leq \left| e^{-Kt} \right| \|(x_b(0) - p_b)\| \leq \|x_b(0) - p_b\|.$$

By the control laws in (10), $\forall t \in T$,

$$\|u_b(t)\| = \| - K(x_b(t) - p_b)\| \leq |K| \|x_b(0) - p_b\|.$$

Since $\|x_b(0) - p_b(0)\| \leq \Delta$, then, $\forall t \in T$,

$$\|u_b(t)\| \leq |K| \|x_b(0) - p_b\| \leq |K|\Delta \leq \frac{u_{max}}{\Delta}\Delta \leq u_{max}.$$

Since u_b is defined over T and $\|u_b(t)\|$ bounded by $u_{max} \; \forall t \in T$, then x_b is Lipschitz continuous. ■

6.1.2 Vertex Addition Control Laws

Once the leader-first-follower pair are assembled, all remaining robots must "attach" to the formation using vertex addition operations. In order to define a unique position that satisfies the geometry defined in $\overline{P}$, we utilize solutions to the circle-circle intersection problem, presented in Chapter 5.3. The following theorem defines control laws that use this estimation.

Theorem 6.2 *For a multi-robot network with proximity range $\Delta \in \mathbb{R}^+$, consider a stable, persistent formation $(\overline{P}, G^\Delta)$, along with three robots i, j, and k. We assume that $u_i(t) = u_j(t) = 0\ \forall t \in T$. We also assume that and $\overline{p}_k = f_k(\overline{p}_i, \overline{p}_j)$ as described in Chapter 5.3. For a maximum velocity $u_{max} \in \mathbb{R}^+$, robot k's control laws are defined by*

$$\hat{p}_k = f_k(x_i, x_j), \tag{13}$$

$$K = \frac{u_{max}}{\Delta}, \tag{14}$$

$$u_k(t) = \begin{cases} -u_{max}\frac{x_k(t)-\hat{p}_k}{\|x_k(t)-\hat{p}_k\|} & \text{if } \|x_k(t) - \hat{p}_k\| > \Delta, \\ -K(x_k(t) - \hat{p}_k) & \text{otherwise.} \end{cases} \tag{15}$$

Then, for every initialization of x_i, x_j, and x_k such that $\|x_k(0) - x_i(0)\| \leq \Delta$, $\|x_k(0) - x_j(0)\| \leq \Delta$, $\|\hat{p}_k - x_i\| \leq \Delta$, $\|\hat{p}_k - x_j\| \leq \Delta$,and $\Im(\hat{p}_k) = 0$,

- *$\dot{\hat{p}}_k(t) = 0\ \forall t \in T$, and $\hat{p}_k$ is a globally, asymptotically stable equilibrium point of p_k,*
- *x_k is continuously differentiable,*
- *$\|x_k(t) - x_i\| \leq \Delta$ and $\|x_k(t) - x_j\| \leq \Delta\ \forall t \in T$,*
- *$\|u_k(t)\| \leq u_{max}\ \forall t \in T$,*
- *x_k is Lipschitz continuous.*

Proof: Since $\dot{x}_i(t) = \dot{x}_j(t) = 0\ \forall t \in T$, this implies that $\dot{\hat{p}}_k(t) = 0\ \forall t \in T$. We define the translated system

$$\hat{x}_k = x_k - \hat{p}_k.$$

Equation (15) implies that

$$\dot{\hat{x}}_k = \dot{x}_k = \begin{cases} -u_{max}\frac{\hat{x}_k}{||\hat{x}_k||} & \text{if } ||\hat{x}_k|| > \Delta, \\ -K\hat{x}_k & \text{otherwise.} \end{cases}$$

The Lyapunov function $V(\hat{x}_k) = \frac{1}{2}\hat{x}_k^T\hat{x}_k$ is globally positive definite. Its time derivative is

$$\begin{aligned} \dot{V}(\hat{x}_k) &= \hat{x}_k^T\dot{\hat{x}}_k \\ &= \begin{cases} -u_{max}||\hat{x}_k|| & \text{if } ||\hat{x}_k|| > \Delta \\ -K||\hat{x}_k||^2 & \text{otherwise.} \end{cases} \end{aligned}$$

Therefore, $\dot{V}(\hat{x}_k)$ is globally negative definite, and the origin of $\hat{x}_k$ is globally asymptotically stable. This implies that $\hat{p}_k$ is a globally, asymptotically stable equilibrium point of x_k.

Here, we show that x_k is continuously differentiable. If $||x_k(0) - \hat{p}_k|| \leq \Delta$, then (15) implies that x_k is continuously differentiable. For the case where $||x_k(0) - \hat{p}_k|| > \Delta$, (15) implies that robot k will have a velocity of magnitude u_{max} in the direction of $\hat{p}_k$ until $||x_k(t) - \hat{p}_k|| = \Delta$, at which point the control laws switch such that $u_k(t) = -K(x_k(t) - \hat{p}_k)$. Let us assume that this switch occurs at $t_\Delta \in T$ such that $||x_k(t_\Delta) - \hat{p}_k|| = \Delta$. This implies that x_k is continuously differentiable over $[0, t_\Delta)$. Note that the limit as $t \to t_\Delta$ from the left is

$$\lim_{t\to t_\Delta^-} u_k(t) = -u_{max}\frac{x_k(t_\Delta) - \hat{p}_k}{||x_k(t_\Delta) - \hat{p}_k||}.$$

At t_Δ, $||x_k(t_\Delta) - \hat{p}_k|| = \Delta$, and the control laws switch such that, $\forall t \in (t_\Delta, \infty)$, $u_k(t) = -K(x_k(t) - \hat{p}_k)$. Hence, x_k is continuously differentiable over (t_Δ, ∞). Also, the limit as $t \to t_\Delta$ from the right is

$$\begin{aligned} \lim_{t\to t_\Delta^+} u_k(t) &= -K(x_k(t_\Delta) - \hat{p}_k) \\ &= -\frac{u_{max}}{\Delta}\frac{x_k(t_\Delta) - \hat{p}_k}{||x_k(t_\Delta) - \hat{p}_k||}||x_k(t_\Delta) - \hat{p}_k|| \\ &= -u_{max}\frac{x_k(t_\Delta) - \hat{p}_k}{||x_k(t_\Delta) - \hat{p}_k||} = \lim_{t\to t_\Delta^-} u_k(t). \end{aligned}$$

Since the limit as $t \to t_\Delta$ is identical from the left and from the right, then u_k is continuous over T, and x_k is continuously differentiable.

To show that $||x_k(t) - x_i|| \leq \Delta$ and $||x_k(t) - x_j|| \leq \Delta\ \forall t \in T$, note that the control laws in (13-15) imply that the range of x_k is in the "straight" line segment parameterized by

$$X_k = \{x_k(0)s + (1-s)\hat{p}_k : s \in (0,1]\}.$$

We define $B_\Delta[x_i]$ and $B_\Delta[x_j]$ as the closed balls of radius Δ around x_i and x_j. By our initialization assumptions, $\{x_k(0), \hat{p}_k\} \subset B_\Delta[x_i] \cap B_\Delta[x_j]$. The convexity of $B_\Delta[x_i] \cap B_\Delta[x_j]$ implies that $X_k \subseteq B_\Delta[x_i] \cap B_\Delta[x_j]$. Hence, $\forall t \in T$, $||x_k(t) - x_i|| \leq \Delta$ and $||x_k(t) - x_j|| \leq \Delta$.

To show that $||u_k(t)|| \leq u_{max}\ \forall t \in T$, note from (15) that, if $||\hat{x}_k(t)|| > \Delta$, then $u_k(t) = -u_{max}\frac{x_k(t)-\hat{x}_k^*}{||x_k(t)-\hat{x}_k^*||}$. This implies that, when $||\hat{x}_k(t)|| > \Delta$,

$$||u_k(t)|| = u_{max}.$$

When $||\hat{x}_k(t)|| \leq \Delta$, $u_k(t) = -K\hat{x}_k(t)$. This implies that

$$||u_k(t)|| = ||-K\hat{x}_k(t)|| \leq |K|\Delta \leq \frac{u_{max}}{\Delta}\Delta \leq u_{max}.$$

Hence, $||u_k(t)|| \leq u_{max}\ \forall t \in T$. Since $\dot{x}_k$ is defined and $||u_b(t)||$ is bounded over T by u_{max} $\forall t \in T$, then x_k is Lipschitz continuous. ∎

Theorems 6.1 and 6.2 specify how robots can satisfy their local geometry, assuming that the robots satisfy the theorems' initialization conditions and have been properly assigned formation positions. We next present an Embedded Graph Grammar (EGG) system that dictates how this assignment occurs, as well as how robots appropriately "switch" control laws when assigned positions. In Chapter 6.3, we also define the initial conditions and controls laws for unassigned robots for guaranteeing that the initialization conditions of Theorems 6.1 and 6.2 are satisfied.

6.2 Embedded Graph Grammars for Formation Assembly

Previously, we defined control laws for controlling the leader-first-follower pair, as well as each vertex addition in the Henneberg sequence to assemble the network graph G^Δ. Here, we present a method to automatically generate *Embedded Graph Grammar (EGG) systems* [41] for assembling desired formations. Through the application of the rules in the rule set, edges may be removed or added to the graph, and the vertex labels may change.

6.2.1 Initial Network Graph

Since the goal of the multi-robot network is to assemble a desired formation, each robot must be assigned a unique position in the formation and attempt to satisfy the local geometry with its immediate successors in the network graph. Note that the network graph G is a *labeled graph* with a *labeling function* $l : \Sigma \mapsto V(G)$. For all $v_i \in V(G)$, this labeling function assigns a label to each vertex v_i corresponding to the *mode* that robot i is in. In this assembly EGG system, the vertex label function l assigns to each vertex in G either a position in $\bar{P}$ or the unassigned label $w \notin \bar{P}$. It also associates a Boolean value to each vertex $v \in V(G)$ depending on whether or not the corresponding robot has reached a location that sufficiently satisfies its constraints with other robots. We use the notation $l(v_i).assign \in \bar{P} \cup \{w\}$ to denote the position in the desired formation that robot i is assigned to, and $l(v_i).final \in \{true, false\}$ as a flag that indicates whether or not robot i has converged sufficiently close to its desired destination. If $l(v_i).assign = w$, we say that robot i is a *wanderer*. Otherwise, we say that robot i is an *assigned* robot.

Initially, all robots in the network are wanderers. Therefore, we model the initial condition for the network graph $G(t)$ as $G(0) = (V, E(0), l_0)$, with $E(0) = \emptyset$, and, $\forall v \in V(G)$, $l_0(v).assign = w$ and $l_0(v).final = false$.

6.2.2 Embedded Graph Grammar Generation

In order to characterize how robots should navigate and establish and maintain constraints with other robots, as well as the corresponding change in network topology, we define graph-transition *rules*. Each rule consists of a vertex-labeled *left graph L* (the input to the rule), a vertex-labeled *right graph R* (the output to the rule), and a *guard* that defines the geometric conditions under which the rule is applicable. These rules define the desired interactions between robots and will be given to each robot, along with the corresponding control laws for each position, in order to execute the formation. Here, we briefly review EGG systems, as discussed in Chapter 2.6.

Assume that $V(L)$ is the vertex set of the left graph L of a rule r. In order for r to be applicable to the robot network modeled by $G(t)$, some subset of $G(t)$ must "look" like L. For this, we follow the notation in [41] and we define a *witness* $h : V(L) \mapsto V(G)$ as a label-preserving isomorphism between the vertices of the left graph L and the vertices of $G(t)$. Witnesses formalize the notion of when two graphs "look" the same (including vertex labels and adjacencies).

However, we also need that certain geometric conditions are satisfied for a rule to be applied. These conditions are encoded through a *guard function* $g : H \times (\mathbb{R}^2 \times \cdots \times \mathbb{R}^2) \mapsto \{true, false\}$, where H is the set of all witnesses for a specific rule. In this work, the guard function is defined to respect the proximity range of the network. This implies that robots must be sufficiently close enough to perceive and communicate with each other in order to apply a rule.

When a witness for a rule exists and the guard evaluates to *true,* we say that the guard is *satisfied* and the rule is *applicable*. If a rule is applicable, the subgraph of $G(t)$ isomorphic to L can be replaced in $G(t)$ by the right graph R in the rule. This can involve both changing labels and/or edges to match the right graph R. A guarded rule is represented by the triple $r = (L \rightharpoonup R, g)$.

As a final building block, each assignment in the vertex labels (i.e. $l(v).assign$) corresponds to a particular control mode based on its position in the desired formation. As such, the robots execute specific control laws for each label.

Depending on the rule set and the network graph, multiple rules may be applicable at the same time. As in [41], we insist that, for a specific time t, if multiple witnesses exist, we allow multiple rules to be applied simultaneously, but insist that the sets of vertices in $V(G)$ that belong to each witness over which the rules are applied must be disjoint. In other

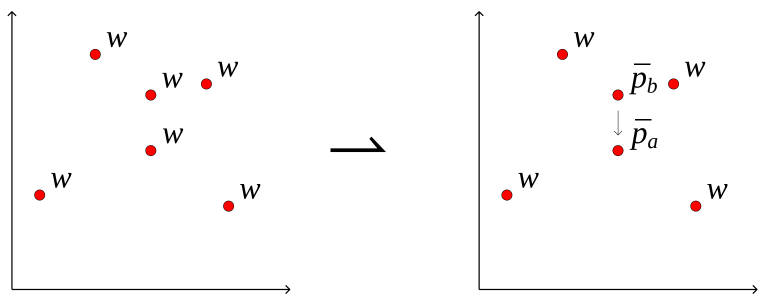


Figure 6.2. Leader-first-follower rules.

words, no vertex in $V(G)$ is allowed to be in two simultaneous rule applications.

Here, we define the specific rules and appropriate control modes that ensure that the desired network graph is achieved. In this work, we augment a traditional EGG system, and describe a *single application EGG system. In this system, we also assume that each rule in the rule set is allowed to be applied once and only once.* This is another condition that is guaranteed by our rule negotiation scheme, which is discussed in Chapter 6.5.

6.2.3 Leader-First-Follower Rules

In Chapter 4.3, we see that the Henneberg sequence begins with a leader-first-follower seed graph G_2. Here, assume that $V(G_2) = \{v_a, v_b\}$, and $E(G_2) = \{(v_b, v_a)\}$. These vertices in $V(G_2)$ correspond to positions $\bar{p}_a$ and $\bar{p}_b$. The robot assigned to $\bar{p}_a$ is the leader, while the robot assigned to $\bar{p}_b$ is the first-follower. Here, we define EGG rules for assigning a pair of robots to these positions and establishing this first constraint. Since we require a unique leader-first-follower seed, we need to make sure that the entire network is involved in the assignment. As such, the leader-first-follower rules involve the entire network. Figure 6.2 describes these leader-first-follower rules.

Leader-First-Follower Position Rule

Through the leader-first-follower seed graph we define a *leader-first-follower position rule* as

$$r^p_{lf} = (L^p_{lf} \rightharpoonup R^p_{lf}, g^p_{lf}),$$

where the left graph is given by the initial network graph $L_{lf}^{p} = G(0)$ and the right graph R_{lf}^{p} is given by

$$V(R_{lf}^{p}) = V(G),$$
$$E(R_{lf}^{p}) = \{(v_2, v_1)\},$$

and, $\forall v \in V(R_{lf}^{p})$,

$$l_{R_{lf}}^{p}(v) = \begin{cases} (\bar{p}_a, true) & \text{if } v = v_1 \\ (\bar{p}_b, false) & \text{if } v = v_2 \\ (w, false) & \text{o.w.} \end{cases}$$

Given a witness h for this rule, the guard g_{lf}^{p} evaluates to true if and only if the robot corresponding to vertex $h(v_1)$ can detect and communicate with each robot in the network. Thus, if $h(v_1) = v_a$, then the guard is true if and only if, $\forall i \in N$, $\|x_a(t) - x_i(t)\| \leq \Delta$. Since the left graph is the initial graph $G(0)$ where each vertex is labeled as a wander and no edges exist, this implies that, initially, any robot within proximity range of all other robots can potentially be a leader, and any robot within proximity range to a potential leader is a potential first-follower.

The control laws associated with labels $(\bar{p}_a, true)$ and $(\bar{p}_b, false)$ are defined in Theorem 6.1. From Theorem 6.1, robot a corresponds to $h(v_1) = v_a$, robot b corresponds to $h(v_2) = v_b$. As such, the leader sets its velocity to zero, while the first-follower begins to approach a distance of d_{ab} from the leader.

Leader-First-Follower Final Rule

As the follower is approaching the its equilibrium point asymptotically, we also have a condition under which we consider the maneuver to be completed. Assume that robot a is assigned $\bar{p}_a$, robot b is assigned $\bar{p}_b$, and p_b is defined as in Theorem 6.1. We define the *leader-first-follower final rule*

$$r_{lf}^{f} = (L_{lf}^{f} \rightharpoonup R_{lf}^{f}, g_{lf}^{f}),$$

whose effect is that the final label of v_b is changed from $false$ to $true$ when the distance between the leader and first-follow has sufficiently converged to p_b for a given threshold value $\epsilon > 0$. In other words, $l(b).final = true$ if and only if $\|x_a(t) - x_b(t)\| \leq \epsilon$. It also sets the control laws of robot b to zero.

6.2.4 Vertex Addition Rules

The leader-first-follower position and final rules specify how we obtain the leader-first-follower seed graph G_2 for the Henneberg sequence. In Chapter 4.3, we also describe how vertex additions generate a graph sequence $(G_2, \ldots, G_n)$ such that $G_n = G^{\Delta}$. Assume that, for graph $G_p : 2 \leq p \leq n$, that $\{v_i, v_j\} \subset V(G_p)$ and a vertex addition operation adds vertex v_k to $V(G_p)$ and edges $\{(v_k, v_i), (v_k, v_j)\}$ to $E(G_p)$ to produce the next graph in the sequence: G_{p+1}. Due to the directions of these edges, the robot corresponding to v_k is in charge of ensuring the proper distance is maintained to the robots corresponding to vertices v_i and v_j, defined by formation positions $\bar{p}_i$, $\bar{p}_j$, and $\bar{p}_k$. This vertex addition operation defines a *vertex addition position rule* as

$$r^p_{va} = (L^p_{va} \rightharpoonup R^p_{va}, g^p_{va}),$$

where the left graph is given by L^p_{va}, with

$$\begin{aligned} V(L^p_{va}) &= \{v_1, v_2, v_3\} \\ E(L^p_{va}) &= \begin{cases} \{(v_1, v_2)\} & \text{if } \exists (v_i, v_j) \in E(G^{\Delta}) \\ \{(v_2, v_1)\} & \text{if } \exists (v_j, v_i) \in E(G^{\Delta}) \\ \emptyset & \text{o.w.} \end{cases} \end{aligned}$$

and, $\forall v \in V(L^p_{va})$,

$$l^p_{L_{va}}(v) = \begin{cases} (\bar{p}_i, true) & \text{if } v = v_1 \\ (\bar{p}_j, true) & \text{if } v = v_2 \\ (w, false) & \text{if } v = v_3. \end{cases}$$

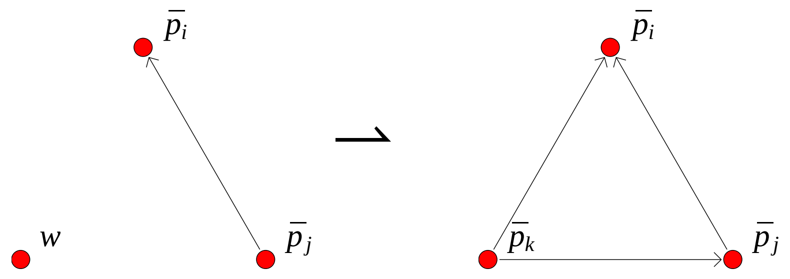


Figure 6.3. Vertex addition rules.

The right graph is given by R^p_{lf}, with

$$V(R^p_{lf}) = \{v_1, v_2, v_3\}$$

$$E(R^p_{lf}) = E(L^p_{va}) \cup \{(v_3, v_1), (v_3, v_2)\}$$

and, $\forall v \in V(R^p_{lf})$,

$$l^p_{R_{va}}(v) = \begin{cases} (\bar{p}_i, true) & \text{if } v = v_1 \\ (\bar{p}_j, true) & \text{if } v = v_2 \\ (\bar{p}_k, false) & \text{if } v = v_3. \end{cases}$$

Given a witness h for this rule, the guard g^p_{va} evaluates to true if an only if the robot corresponding to vertex $h(v_3)$ is close enough to $h(v_1)$ and $h(v_2)$ to be able to detect them. In other words, assuming that $h(v_1) = v_i$, $h(v_2) = v_j$, and $h(v_3) = v_k$, the guard is true at time t if and only if $\|x_k(t) - x_i(t)\| \leq \Delta$ and $\|x_k(t) - x_j(t)\| \leq \Delta$. The control laws associated with labels $(\bar{p}_i, true)$ and $(\bar{p}_j, true)$ and $(\bar{p}_k, false)$ are defined in Theorem 6.2. As such, the robots assigned $\bar{p}_i$ and $\bar{p}_j$ have zero velocity, while the robot assigned $\bar{p}_k$ begins to move towards its estimate of its desired position. Figure 6.3 shows vertex addition rules.

Vertex Addition Final Rule

As robot k is approaching its equilibrium point asymptotically, we also have a condition under which we consider the maneuver to be completed. For this we define the *vertex addition final rule*

$$r^f_{va} = (L^f_{va} \rightharpoonup R^f_{va}, g^f_{va}),$$

whose only effect is that the label at vertex $h(v_3) = k$ is changed from $false$ to *true* when $\|\hat{p}_k(t) - x_k(t)\| < \epsilon$, for a given threshold value $\epsilon > 0$. Like the leader-first-follower final rule, it also sets u_k to zero.

Based on the EGG rules we have defined, the robots switch to the control modes defined in Chapter 6.1 and then switch out of them after sufficient convergence to their estimated desired states, setting their control laws to zero. This implies that there will be some final error in the formation, and the amount of this error is intuitively determined by how small we define ϵ. In Chapter 6.4, we discuss this formation error, and show that it is, in fact, bounded by our choice of ϵ. By bounding ϵ to be arbitrarily small, we can also arbitrarily bound the resulting formation error to be arbitrarily small. In Chapter 6.3, we discuss the implementation of "wander mode" to ensure that the rule guards are satisfied, and that the unassigned robots (the "wanderers") will satisfy the guards of the vertex additions as the formation expands from the network's initial state.

6.3 Wander Mode

Here, we present *wander mode*, which allows unassigned wanderers to satisfy the guards of available vertex addition rules. The wanderers must navigate to positions in the network where vertex addition rules are applicable. However, each robot is limited to perceiving only the portions of the network that are within its proximity range. At any given time, the locations in the network where vertex additions are applicable may be outside the proximity range of any wanderer. We need a way for the wanderers to navigate without global information.

To implement wander mode, each robot assigned a position is given a *hop-counter* λ, which it communicates to all assigned robots within proximity range. We set to zero the hop-counters of all assigned robots whose labels allow them to participate in vertex addition position rules that have not been applied. This implies that their labels occur in the left graph of position rules that have not yet been applied (This also requires robots to "keep track" of what rules have and have not been applied). Assume that robot i is an assigned robot that cannot participate in a vertex addition position rule. Assume that Λ_i is the set of all the hop-counters of all robots within proximity range of i. Then we define robot i's hop-counter by

$$\lambda_i = \begin{cases} \min(\Lambda_i) + 1 & \text{if } \min(\Lambda_i) < n \\ n & \text{otherwise.} \end{cases} \tag{16}$$

Since all robots with hop-counters equal to zero have been assigned positions in the desired formation, and since each edge in G^Δ has a length less than or equal to Δ, this implies that there always exists a "path" of assigned robots with decreasing hop-counters that leads to a hop-counter of zero, if such a robot exists. Wander mode is defined so that wanderers perform circular motion around the robot with the lowest hop-counter in proximity range. When a robot with a lower hop-counter comes into its perception, it switches to circle around that robot. This process repeats until the wanderer finds an assigned robot with a hop-counter equal to zero.

Once a wanderer encounters an robot whose hop counter equals zero, it enters an exclusive partnering relationship with that robot. The assigned partner i changes its hop-counter from zero to $\min(\Lambda_i) + 1$. The assigned partner also refuses any more partnerships with other wanderers. As each rule is applied, the robots involved in the rule application reevaluate their hop-counters as defined in (16).

Note that each vertex addition rule has two vertices with labels that assign a hop-counter of zero to assigned robots. This implies that two wanderers can potentially be partnered with different robots, but for the same rule. Therefore, if the hop-counter changes from zero to another value, this signals any partnered wanderers to abandon the partnership and to follow a path to another robot with a zero hop-counter. Since this situation only occurs when all robots required for a vertex addition rule are present, then this implies that the redundant partnered wanderer is always freed, and can proceed towards another vertex addition rule opportunity.

The definition of hop-counters also implies that, when all robots that can participate in vertex additions have partners, there may be intervals of time where there is no hop-counter equal to zero. However, this situation guarantees that a vertex addition rule is applied, since all assigned robots that can participate in vertex additions have a partnered wanderer. Figure 6.4 shows a wanderer performing wander mode.

The following control laws are performed by the wanderers. For a wanderer robot *w* and an assigned robot *a*, these control laws stabilize the wanderer to a circular velocity around robot *a* at a distance approaching Δ as $t \to \infty$.

Theorem 6.3 *Consider a stably, persistently feasible formation defined by $\overline{P}$ and corresponding stable, persistent network graph G, along with a pair of robots a and w. For a proximity range $\Delta \in \mathbb{R}^+$ and a maximum velocity $u_{max} \in \mathbb{R}^+$, robot w's control laws are defined by*

$$x_w^* = \Delta \frac{x_w - x_a}{||x_w - x_a||} + x_a, \tag{17}$$

$$K_w = \frac{u_{max}}{2\Delta}, \tag{18}$$

$$Q_1 = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}, \tag{19}$$

$$\begin{aligned} u_w &= \frac{u_{max}}{2} Q_1 \frac{x_w - x_a}{||x_w - x_a||} - K_w (x_w - x_w^*) \\ &= \frac{u_{max}}{2} Q_1 \frac{x_w - x_a}{||x_w - x_a||} - K_w \left(1 - \frac{\Delta}{||x_w - x_a||}\right)(x_w - x_a). \end{aligned} \tag{20}$$

We assume that $u_a(t) = 0\ \forall t \in T$ (i.e., $x_a(t)$ is a constant $\forall t \in T$). Then, for every initialization of the pair such that $0 < ||x_w(0) - x_a(0)|| \leq \Delta$,

- x_w *is continuously differentiable,*
- $\lim_{t\to\infty} ||x_w(t) - x_a|| = \Delta$,
- $\lim_{t\to\infty} \left(u_w(t) - \frac{u_{max}}{2} Q_1 \frac{x_w - x_a}{||x_w - x_a||}\right) = 0$,
- $||x_w(t) - x_a|| \leq \Delta\ \forall t \in T$,

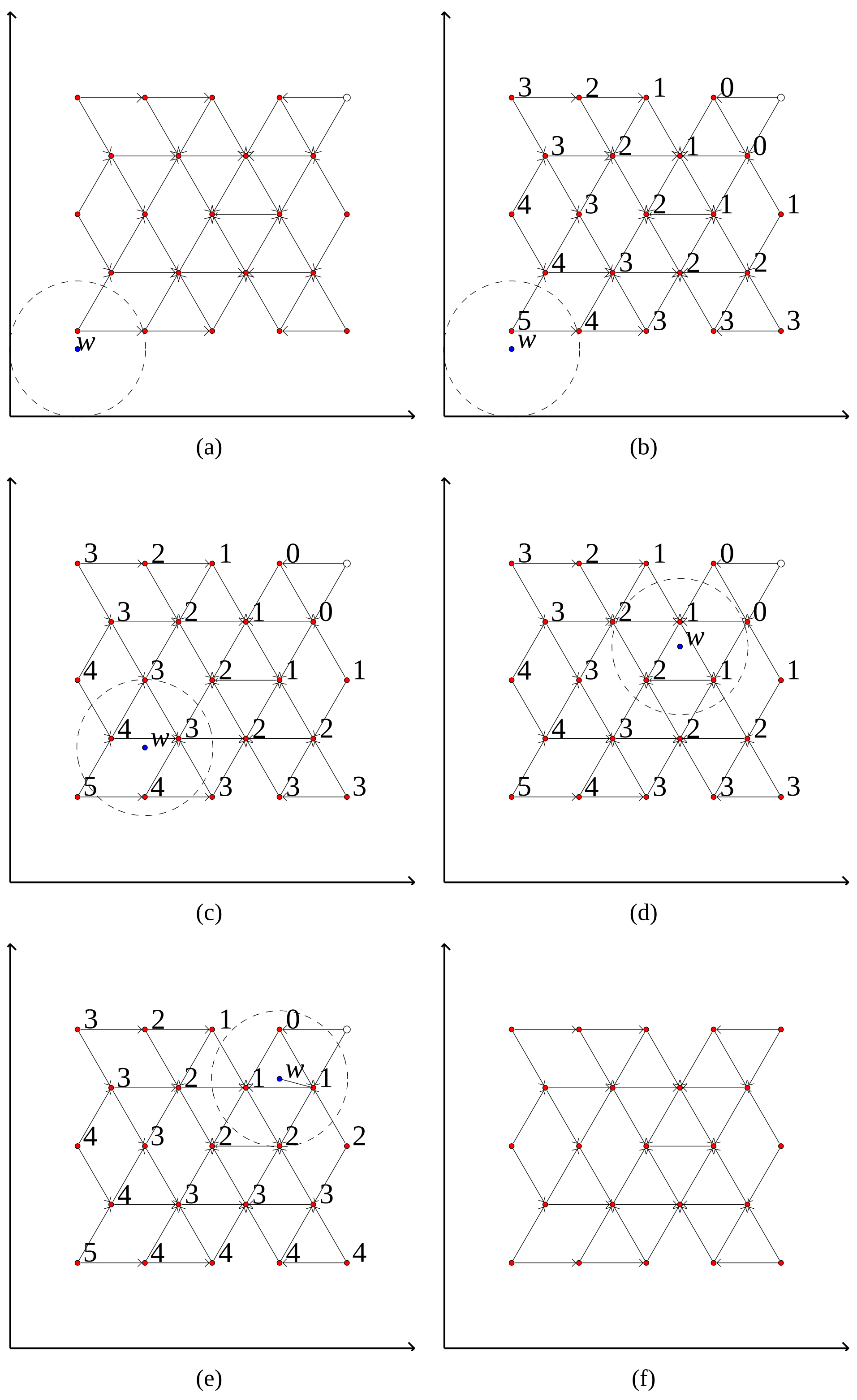


Figure 6.4. Wander mode. Assigned robots establish *hop-counters*, which count the "hops" to an available vertex addition. By sharing these hop-counters with wanderers within proximity range, wanderers can perform circular motion and follow the hops to available vertex additions.

- $\|u_w(t)\| \leq u_{max}$ $\forall t \in T$, *and*
- x_w *is Lipschitz continuous.*

Proof: First, we must show that $x_w(t) \neq x_a$ $\forall t \in T$. To analyze the behavior of $\|x_w - x_a\|$, note that

$$\frac{d}{dt}\left(\frac{1}{2}\|x_w(t) - x_a\|^2\right) = (x_w(t) - x_a)^T(\dot{x}_w(t) - \dot{x}_a)$$

$$= (x_w(t) - x_a)^T \dot{x}_w(t).$$

Since Q_1 is an rotation matrix corresponding to a rotation of $\frac{\pi}{2}$, then, $\forall x \in \mathbb{R}^2$, $Q_1 x$ is orthogonal to x, and $x^T Q_1 x = 0$. Then, substituting (20) into the previous yields

$$\frac{d}{dt}\left(\frac{1}{2}\|x_w(t) - x_a\|^2\right) = -K_w\left(1 - \frac{\Delta}{\|x_w(t) - x_a\|}\right)\|x_w(t) - x_a\|^2. \tag{21}$$

This implies that $\frac{d}{dt}\left(\frac{1}{2}\|x_w(t) - x_a(t)\|^2\right) > 0$ if $\|x_w(t) - x_a(t)\| < \Delta$. In other words, the distance between robots w and a is always increasing if their distance is less than Δ.

By our assumptions, $x_w(0) \neq x_a$. This implies that $u_w(0)$ is defined. If we assume that, for some $t > 0$, that $x_w(t) = x_a$, we always have a contradiction. This assumption implies that there exists $t_1 > 0$ such that $t_1 < t$, $\frac{d}{dt}\left(\frac{1}{2}\|x_w(t) - x_a\|^2\right) < 0$, and $\|x_w(t) - x_a(t)\| < \Delta$. In other words, this implies that there exists a time before t such that the distance between robots w and a is decreasing and their distance is less than Δ. This violates (21). Hence, $x_w(t) \neq x_a$ $\forall t \in T$, and x_w is continuously differentiable.

To show that $\lim_{t\to\infty}\|x_w(t) - x_a\| = \Delta$, we first show the stability of the origin of $\tilde{x}_w = x_w - x_w^*$. By the quotient rule, we have that

$$\dot{x}_w^* = \frac{\Delta}{\|x_w - x_a\|^2}\left(\|x_w - x_a\| - \frac{(x_w - x_a)(x_w - x_a)^T}{\|x_w - x_a\|}\right)\dot{x}_w.$$

Substituting (20) into the previous yields

$$\dot{x}_w^* = \frac{\Delta}{\|x_w - x_a\|}\frac{u_{max}}{2}Q_1\frac{x_w - x_a}{\|x_w - x_a\|}. \tag{22}$$

The Lyapunov function $V(\tilde{x}_w) = \frac{1}{2}\tilde{x}_w^T\tilde{x}_w$ is positive definite. Also, from (20) and (22),

$$\begin{aligned}\dot{V}(\tilde{x}_w) &= \tilde{x}_w^T\dot{\tilde{x}}_w \\ &= \tilde{x}_w^T(\dot{x}_w - \dot{x}_w^*) \\ &= \tilde{x}_w^T\left(\left(1 - \frac{\Delta}{\|x_w - x_a\|}\right)\frac{u_{max}}{2}Q_1\frac{x_w - x_a}{\|x_w - x_a\|}\right. \\ &\quad \left. - K_w\tilde{x}_w\right)\end{aligned}$$

Since $\tilde{x}_w$ and $(x_w - x_a)$ are parallel, then $\tilde{x}_w^T Q_1(x_w - x_a) = 0$. Hence,

$$\dot{V}(\tilde{x}_w) = -K_w\tilde{x}_w^T\tilde{x}_w = -K_w\|\tilde{x}_w\|^2.$$

Thus, $\dot{V}(\tilde{x}_w)$ is negative definite. This implies that the origin of $\tilde{x}_w$ is globally, asymptotically stable. Therefore, x_w^* is a globally, asymptotically stable equilibrium point of x_w, and $\lim_{t\to\infty}\|x_w(t) - x_a\| = \lim_{t\to\infty}\|x_w^*(t) - x_a\| = \Delta$.

Since

$$\lim_{\|x_w(t)-x_a\|\to\Delta} u_w(t) = \frac{u_{max}}{2}Q_1\frac{x_w - x_a}{\|x_w - x_a\|},$$

then $\lim_{t\to\infty}\left(u_w(t) - \frac{u_{max}}{2}Q_1\frac{x_w - x_a}{\|x_w - x_a\|}\right) = 0$. Hence, as $t \to \infty$, the wander's velocity approaches a circular velocity around robot a with radius of Δ.

To see that $\|x_w(t) - x_a\| \leq \Delta\ \forall t \in T$, note from (21) that $\frac{d}{dt}\left(\frac{1}{2}\|x_w(t) - x_a(t)\|^2\right) < 0$ if $\|x_w(t) - x_a(t)\| > \Delta$. In other words, the distance between robot w and robot a is always decreasing if their distance is greater than Δ. If we assume that, for some $t \in T$, that $\|x_w(t) - x_a\| > \Delta$, we always have a contradiction. If $t = 0$, then this contradicts are initialization assumptions. If $t > 0$, then this implies that there exists a $t_1 > 0$ such that $t_1 \leq t$, $\|x_w(t_1) - x_a\| > \Delta$, and $\frac{d}{dt}\left(\frac{1}{2}\|x_w(t_1) - x_a\|^2\right) > 0$. In other words, this assumption implies that there is a time before t such that the distance between robots a and w is increasing and their distance is greater than Δ. This violates (21). This contradiction shows us that no such t exists. Hence, $\|x_w(t) - x_a\| \leq \Delta\ \forall t \in T$.

To show that $\|u_w(t)\| \leq u_{max}\ \forall t \in T$, (20) implies that

$$\|u_w(t)\| \leq \left\|\frac{u_{max}}{2}\right\| + |K_w|\|x_w(t) - x_w^*(t)\|.$$

Since $\frac{d}{dt}\|x_w(t)-x_w^*(t)\|^2 < 0\ \forall t \in T$ then $\|x_w(t)-x_w^*(t)\| \leq \|x_w(0)-x_w^*(0)\|\ \forall t \in T$. From (17), the points x_a, $x_w(0)$, and $x_w^*(0)$ are collinear, $\|x_w^*(0) - x_a\| = \Delta$, and x_a cannot be between $x_w(0)$ and $x_w^*(0)$. Since $\|x_w(0) - x_a\| \leq \Delta$, this implies that $\|x_w(0) - x_w^*(0)\| \leq \Delta$. Hence,

$$\left\|\frac{u_{max}}{2}\right\| + |K_w|\|x_w(t) - x_w^*(t)\| \leq \frac{u_{max}}{2} + \frac{u_{max}}{2\Delta}\Delta \leq u_{max},$$

and $\|u_w(t)\| \leq u_{max}\ \forall t \in T$. Since u_w is defined and bounded by u_{max} over all T, then x_w is Lipschitz continuous. ∎

The following corollary shows that, while circling a robot in wander mode, a wanderer will always satisfy the guard of any available vertex addition rules.

Corollary 6.4 *Consider a trio of robots a, b, and w. Assume that robot w is executing the wander mode control laws as in Theorem 6.3, performing circular motion around robot a. We define $B_\Delta(x_a) \cap B_\Delta(x_b)$ as the intersections of the* open *balls of radius Δ around x_a and x_b, and let $B_\Delta[x_a] \cap B_\Delta[x_b]$ be the intersections of the corresponding* closed *balls. For any initialization of the trio such that $B_\Delta(x_a) \cap B_\Delta(x_b) \neq \emptyset$, there exists a time $t \in T$ such that $x_w(t) \in B_\Delta[x_a] \cap B_\Delta[x_b]$.*

Proof: Since $B_\Delta(x_a) \cap B_\Delta(x_b) \neq \emptyset$, then $\|x_a - x_b\| < 2\Delta$. From Theorem 6.3, there exists a time $t \in T$ such that x_a, x_b, and x_w are collinear, and

$$\begin{aligned} x_w^* &= \Delta\frac{x_w - x_a}{\|x_w - x_a\|} + x_a \\ &= \Delta\frac{x_b - x_a}{\|x_b - x_a\|} + x_a. \end{aligned}$$

This occurs when x_w is in the direction of x_b from x_a. Since $\|x_w^*-x_a\| = \Delta$ and $\|x_a-x_b\| < 2\Delta$, then $\|x_w^* - x_b\| < \Delta$ and $x_w^* \in B_\Delta(x_b)$. Since the origin of $\tilde{x}_w = x_w - x_w^*$ is globally, asymptotically stable, this implies that there exists a time t such that $\|\tilde{x}_w(t)\|$ is arbitrarily small. Hence, there exists a time t when $x_w(t) \in B_\Delta(x_b)$. Since Theorem 6.3 shows that $x_w \in B_\Delta[x_a]\ \forall t \in T$, this implies that there exists a $t \in T$ such that $x_w(t) \in B_\Delta[x_a] \cap B_\Delta[x_b]$.

∎

6.4 Formation Error

Here, we define the formation error. We also show that we can arbitrarily bound the resulting formation error.

By the control laws in Theorem 6.1, the robot assigned the leader position does not move. Therefore, the position of the leader establishes the translation of the formation $\overline{P}$ to the robots' environment. The control laws in Theorem 6.1 also show that the initial position of the first-follower establishes the rotation of the formation $\overline{P}$ to the robots' environment.

To define the network's formation error, we assume that, for a formation $\overline{P}$, the appropriate EGG system is generated and executed such that each rule is applied once, and that, at time $t \in T$, each robot has completed the execution of its control laws. The execution of the EGG system does not insist that any particular robot is assigned any specific position in the formation. For notational clarity, we assume that the robots and positions are indexed such that, $\forall i \in N$, robot i is assigned position $\overline{p_i}$ such that $l(v_i) = \overline{p_i}$. If $\overline{p_a}$ and $\overline{p_b}$ describe the leader and first-follower positions in $\overline{P}$, then robots a and b are assigned the leader and first-follower positions, respectively, as described in Theorem 6.1.

Recall that the desired formation $\overline{P}$ does not necessarily specify the desired location of the network in the environment. Rather, the positions of the formation define a relative geometry that we want the network to satisfy. However, to further aid the definition of the network's formation error, let us assume without loss of generality that the leader robot a is at location $x_a = \overline{p_a}$, and that the first-follower robot b is navigating towards position $p_b = \overline{p_b}$ as defined in Theorem 6.1. Then, $\forall i \in N$, $\overline{p_i}$ does, in fact, define the location of zero *formation error* for robot i. This allows us to define the formation error of each robot such that, $\forall i \in N$, $\tilde{x}_i = \overline{p_i} - x_i$ is the formation error of robot i.

By this definition of formation error, we assume that the leader always has zero formation error (i.e., $x_a(t) = \overline{p_a}\ \forall t \in T$). Similarly, the first-follower executing the control laws in 6.1 is always navigating to a position of zero formation error (i.e., $p_b(t) = \overline{p_b}\ \forall t \in T$). However, the first-follower is exponentially approaching $\overline{p_b}$; it never achieves it. In fact, based on our EGG system definition, the first-follower stops when it is within ϵ of $\bar{p}_b$. The remaining robots must *estimate* their zero formation error positions based on the positions of other robots, as described in Theorem 6.2. The following theorem shows that we can arbitrarily bound the final formation error by choosing an appropriately small epsilon.

Theorem 6.5 *Consider a multi-robot network with proximity range $\Delta \in \mathbb{R}^+$, stably, persistently feasible formation $\bar{P}$ and stable, minimally persistent network graph G. Assume that the EGG system described in Chapter 6.2 is implemented, and the guards of each rule are satisfied such that each rule is applied in the network. This implies that, $\forall i \in N$,*

$$\lim_{\epsilon \to 0} \|\tilde{x}_i\| = 0.$$

Proof: For convenience, let us assume that the vertices in G are indexed such that robot 1 is leader, robot 2 is the first-follower, and, $\forall (v_j, v_i) \in E(G),\ j > i$. This is always possible since stably, persistent formation graphs are always directed acyclic graphs, as discussed in Chapter 4.4.

Since we assume that $\bar{p}_1 = x_1$, this implies that $\|\tilde{x}_1\| = 0\ \forall \epsilon \in \mathbb{R}^+$. Note that the first follower stops such that $\|\tilde{x}_2\| = \|x_2 - \bar{p}_2\| = \epsilon$. Since $\lim_{\epsilon \to 0} \epsilon = 0$, this implies that $\lim_{\epsilon \to 0} \|\tilde{x}_2\| = 0$.

Note that all additional robots are added to the formation by vertex additions. If $n \geq 3$, then $\{(v_3, v_1), (v_3, v_2)\} \subset E(G)$, and the leader and first-follower robots are immediate successors of robot 3. Robot 3 will drive to a point within ϵ of its *estimation* of $\hat{p}_3$ such that $\hat{p}_3 = f_3(x_1, x_2) = f_3(\bar{p}_1 + \tilde{x}_1, \bar{p}_2 + \tilde{x}_2)$. Since $\bar{p}_3 = f_3(\bar{p}_1, \bar{p}_2)$, then this implies that

$$\lim_{\tilde{x}_1 \to 0, \tilde{x}_2 \to 0} \|\hat{p}_3 - \bar{p}_3\| = \|f_3(\bar{p}_1 + 0, \bar{p}_2 + 0) - f_3(\bar{p}_1, \bar{p}_2)\| = 0.$$

This implies that $\lim_{\epsilon \to 0} \|\hat{p}_3 - \bar{p}_3\| = 0$. Since robot 3 drives within ϵ of $\hat{p}_3$ before stopping, this implies that $\lim_{\epsilon \to 0} \|x_3 - \hat{p}_3\| = 0$. Hence,

$$\lim_{\epsilon \to 0} \|\tilde{x}_3\| = \lim_{\epsilon \to 0} \|x_3 - \bar{p}_3\| = \lim_{\epsilon \to 0} \|x_3 - \hat{p}_3 + \hat{p}_3 - \bar{p}_3\| \leq \lim_{\epsilon \to 0} \|x_3 - \hat{p}_3\| + \|\hat{p}_3 - \bar{p}_3\| = 0.$$

Since 0 is the lower bound of $||\tilde{x}_3||$, this implies that $\lim_{\epsilon\to 0} ||\tilde{x}_3|| = 0$. Thus,

$$\lim_{\epsilon\to 0} ||\tilde{x}_1|| = \lim_{\epsilon\to 0} ||\tilde{x}_2|| = \lim_{\epsilon\to 0} ||\tilde{x}_3|| = 0.$$

Remember that the formation graph is an acyclic graph, and that we have indexed our indices where 1 is the leader, 2 is the first-follower, and, $\forall k \in N : k \geq 3$, robot k attaches with a vertex addition. Any robot attaching with a vertex addition will do so with robots i and j such that i and j are less than k. Then, based on our previous results and induction, $\forall \{k \in N : k \geq 3\}$, $\exists (v_i, v_j) \in V(G) \times V(G)$ such that $\{(v_k, v_i), (v_k, v_j)\} \subset E(G)$, $k > i$, and $k > j$. This implies that

$$\lim_{\epsilon\to 0} ||\tilde{x}_i|| = \lim_{\epsilon\to 0} ||\tilde{x}_j|| = 0.$$

Therefore, $\lim_{\epsilon\to 0} ||\hat{p}_k - \overline{p}_k|| = 0$, and $\lim_{\epsilon\to 0} ||\tilde{x}_k|| = 0$. Hence, $\lim_{\epsilon\to 0} ||\tilde{x}_i|| = 0 \ \forall i \in N$. ■

6.5 Embedded Graph Grammar Implementation

Having automatically generated the set of rules and control modes defining the EGG as discussed in Chapter 6.2, this EGG is then given to the robots, whose task is to execute them. As discussed in this chapter, as well as Chapter 2.6, our EGG system requires that no robots are involved in simultaneous rule applications, and that robots keep track of which rules have been applied. Here, we describe how this is accomplished with a decentralized network. Specifically, we describe how robots negotiate rule applications and keep track of which rules have been applied.

6.5.1 Primaries and Rule Evaluation

Since this is a decentralized network, the robots must communicate and negotiate rule applications in a manner consistent with the EGG defined in 6.2. The label and adjacency information is distributed across the network such that each robot has immediate access only to its own label and adjacency information. The label and adjacency information corresponding to other robots can only be obtained through wireless communication. For the EGG to be successfully executed with the network, the robots in the network must change modes and execute control laws in a manner defined by the EGG's guarded rules, labels, and the corresponding control law for each label. For the EGG we have defined, this also requires the network to guarantee that no rule is applied more than once.

For each rule $r = (L \rightharpoonup R, g)$, there is a vertex $v \in V(L)$ that is defined as the *primary vertex* of the rule. In each rule, the primary vertex corresponds to the robot in the rule application that is within proximity range of every other robot involved in the rule application. Therefore, the primary robot has enough local graph information to apply the rule. For leader-first-follower position rules, this is the vertex corresponding to the leader robot. For vertex addition position rules, this is the vertex corresponding to the wanderer in the left graph. For final rules, this is the vertex with the $false$ final label.

When a witness exists that maps a rule's primary vertex to a robot's vertex in $G(t)$, we say that the robot is a *primary robot*. Robots only attempt to apply rules with witnesses that map themselves to the primaries of the rules. Each robot determines whether or not it is a primary by examining its label, the rule set, requesting the graph information of other robots, and comparing it to the left graphs of the rules to see if one is applicable. If so, then the primary robots attempt to apply the rules to the network graph by modifying the local graph information of itself and its neighbors. This process is called *rule evaluation*.

Since a witness for a rule is a label-preserving graph isomorphism from the vertices of the left graph of a rule to the vertices of $G(t)$, this seems to suggest a potentially expensive computation, especially when considering an exhaustive, global search for all witnesses. However, the definition of these rules and the decentralized nature of the network greatly reduce this complexity.

The leader-first-follower position rule is a large rule, in that every vertex in $G(t)$ is included in the rule's left graph. At first glance, this may suggest checking every permutation of the vertices in $G(t)$ for a label-preserving isomorphism. However, since the left graph of this rule is the initial graph $G(0)$, every permutation of the vertices in $G(t)$ is a label-preserving graph isomorphism at time $t = 0$, and any permutation of rules checked will imply that the rule is applicable. Since this rule is applied only once and at the beginning of execution, exhaustive searches for graph isomorphisms for this rule are never performed.

When evaluating a vertex addition rule, a wanderer is searching for two assigned robots with the labels corresponding to the assigned labels in the left graph. Note that a vertex addition rule adds all the edges from the new vertex to the vertices in $G(t)$ that correspond to the edges in G^{Δ}. Therefore, if the wanderer finds two robots whose labels match the left graph of a vertex addition rule, then the edges must also satisfy the rule, implying that the rule is applicable. This reduces the search for a vertex addition rule witness to a search for two robots with specific labels, which can be solved in linear time ($O(n)$).

6.5.2 Prioritized Lock Negotiation

It is necessary that each primary robot has exclusive control of all robots necessary to apply a rule; if not, then it is possible for multiple primary robots to modify the graph information in a manner inconsistent with the rules, or apply a rule more than once, producing *graph inconsistencies*. Graph inconsistencies occur when there exists subgraphs of $G(t)$ that are not intended to exist by the EGG design. For example, assume that two wanderers simultaneously apply the same vertex addition position rule, assigning both robots the same position in the desired formation. This would result in a redundant robot for one position, and a missing robot for another position.

In Chapter 3.2, we presented a communication protocol for networks with global communication. However, the network we consider here does not have global communication ability. Therefore, we present a communication scheme for preventing graph inconsistencies that is applicable for networks with only local communication abilities. To this end, we define the *prioritized lock negotiation* communication scheme, shown in Figure 6.5.

The prioritized lock negotiation communication scheme gives primaries exclusive control of other robots' EGG information through a series of *lock negotiations*. When a primary robot wants to apply a rule involving another robot, it performs a *lock request* for that robot. We define a *locked robot* as a robot that is locked by a primary. We define an

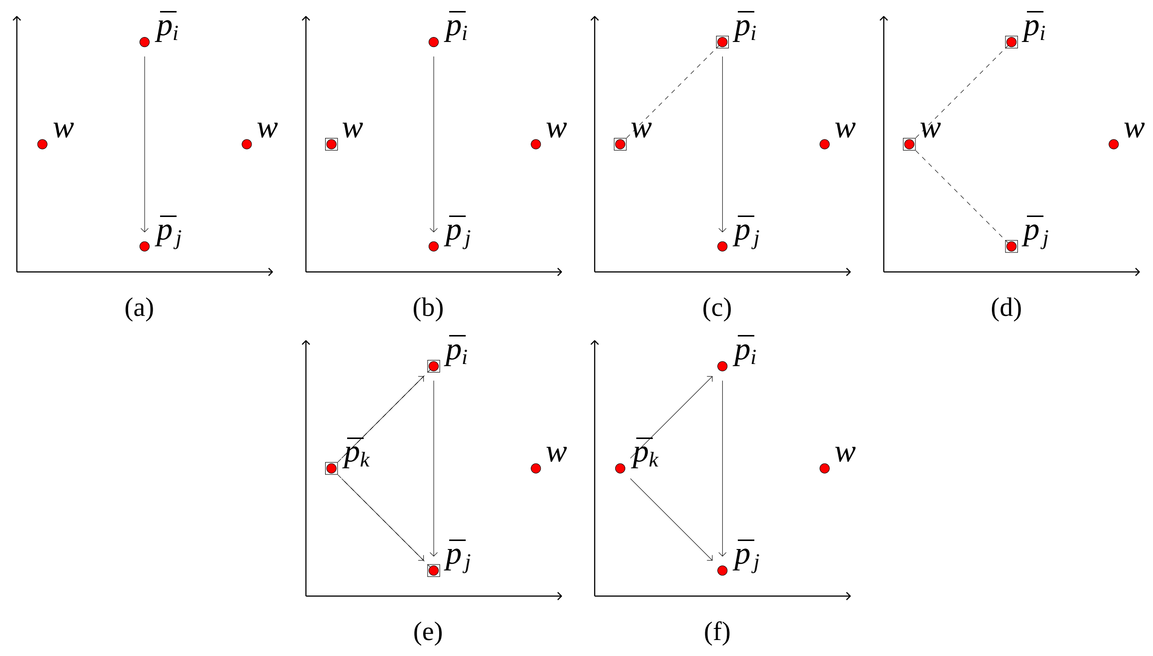


Figure 6.5. Prioritized lock negotiation. When a wander wants to apply a rule, it first acquires the locks to robots involved, including itself. In this figure, locks are indicated by rectangles around the robots' states, and dotted lines indicate which wanderer owns the lock. When a robot owns the lock of another robot, it has exclusive control of its labeled graph information. Once the wander has all the locks, it applies the rule. Once applied, the locks are released.

unlocked robot as a robot that is not locked by a primary. If the robot being requested for a lock is unlocked, it accepts the lock of the primary and records the primary's index. We say that the primary *owns* the lock on this robot. The locked robot will refuse any lock requests while locked. Once locked, the locked robot only allows the owner of its lock to modify the locked robot's EGG information.

Once a primary has locked the entire set of robots necessary for the rule application (including itself), it verifies that the rule is still applicable, i.e. the graph information is still consistent with the rule, and the rule has not been applied. Since each robot has a copy of the rule set, we exploit the locality of the guarded rules to prevent any rule from being applied more than once. As each rule is applied, it is removed from the rule sets of the robots involved in the application. Also, before a primary can apply a rule with its locked robots, it must first verify that each locked robot has the rule in its rule set.

From the definition of the leader-first-follower position rule, each robot in the network participates in the rule application. Therefore, since each robot removes the leader-first-follower positions rules from their set of rules as it is applied, this guarantees that the leader-first-follower position rule is applied only once. Similarly, when vertex addition rules are applied, the corresponding vertex addition position rule is removed from the rule sets of the involved robots. However, this does not encompass the entire network of robots. After a vertex addition rule is applied, the remaining wander robots (which make up the primaries in vertex additions) are not aware of previously applied vertex additions.

Since applied rules are removed from the rule sets of the involved robots, the uniqueness of the leader-first-follower subgraph implies that all vertex addition rules that apply with the leader and follower are applied only once and result in unique subgraphs. By induction, this comparison of rule sets implies that all vertex additions require a unique subgraph of $G(t)$, and this prevents all rules from being applied more than once, guaranteeing the uniqueness of all positions assigned by position rules. This verification also allows wanderers to remove rules that have been applied from their rule sets, preventing redundant rule evaluations. When the primary has completed all modifications of graph information necessary to apply the rule, including the removal of the applied rule from the rule sets of the involved robots, it then *unlocks* all the robots it has locked.

This system of lock negotiations ensures that graph inconsistencies are avoided and that rules are not applied more than once. However, the rule set could define many witnesses, and, therefore, many primary robots. With many primary robots attempting to lock sets of other robots, it is possible for primary robots to lock robots in a manner that prevents any applicable rule from being applied. We define this as *deadlock*.

To prevent deadlock, we define a priority to each robot that corresponds to its index. We say that robot j has a higher priority than robot i if $i < j$. Since each robot has a unique index, no robots have the same priority. When a locked robot refuses a lock request, it communicates the index of the primary that locked it to the robot requesting the lock. Based on this index, the requesting primary robot does either one of two things:

1. If it has a higher priority than the robot that owns the lock, it immediately retries the lock request.

2. If it has a lower priority than the robot that owns the lock, it immediately unlocks all robots that it owns locks for, and waits for a time τ before reattempting the rule.

We define τ as a worst-case period of time long enough to allow n robots to attempt n rule negotiations in series. Therefore, when robots compete for locks, there will always be a lowest priority robot. This implies that, if no robot can acquire a lock to all the robots involved in a rule application, then the lowest priority robot will always release its locks and wait for time τ before trying again. There is also always a highest priority robot, who does not release its locks, and does not delay in reattempting to acquire a lock. Even if more robots begin attempting to compete for the same locks, the delay time τ is defined for a worst-case, meaning that there is time for n rule negotiations, and subsequently n more robots to "drop out", releasing its locks and delaying for their own τ, before the delayed robot attempts again. This implies that, in a worst-case scenario, there will, at some time, be only one robot attempting to acquire locks. Therefore, the network cannot be constantly deadlocked.

6.6 Simulation Results

Here, we present simulation results of our method of formation assembly. We consider a group of robots and a desired triangulation formation that corresponds to a coverage pattern.

6.6.1 Triangulation Scenario

To demonstrate the assembly of formations with the multi-robot network, we implement the following scenario: We have a network of $n \geq 2$ robots with data collection sensors, and we wish to distribute them in a triangular coverage pattern over an area of interest. Triangular coverage patterns occur frequently, since they dictate an equal distance between

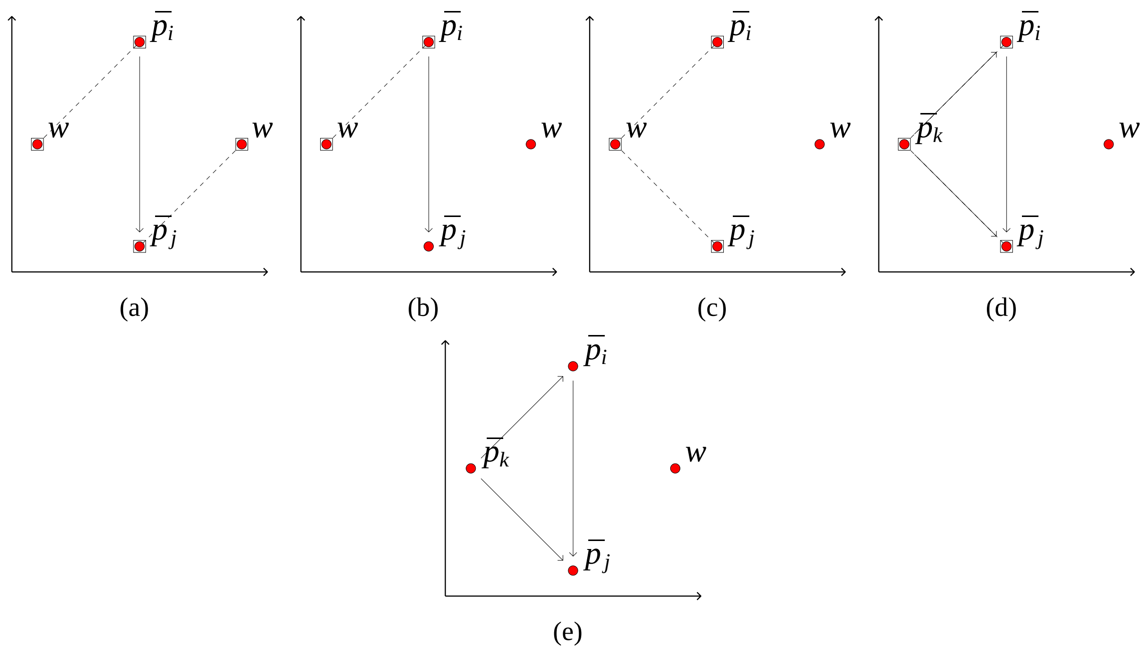


Figure 6.6. When two wanderers compete for the same locks, deadlock can occur. We assign a unique priority to each robot; in this case, the wander on the left has highest priority. When the robot on the right encounters the lock owned by the higher priority wanderer, it releases its locks. This allows the higher priority robot to acquire its locks and apply the rule.

each adjacent robot in the coverage pattern. Therefore, we enter a triangulation pattern of positions in our graphical program shown in Figure 6.1.

In Chapter 4.4, a method for automatically generating stable, persistent graphs for desired formations is presented. This is used to define the minimally persistent formation graph G^{Δ}, as well as a leader-first-follower seed G_2, and a sequence of vertex addition operations that assemble G^{Δ} from G_2.

Using the methods previously described, the EGG defined by G^{Δ} and the desired formation $\bar{P}$ is automatically generated. The EGG is then given to each robot, and the network is positioned in the area of interest. Then the EGG is executed.

6.6.2 Triangulation Simulation

Figure 6.7 shows a simulation of these results with $n = 23$ robots, using a triangulation pattern with 20 m constraints. In this simulation, a proximity range of $\Delta = 21$ m is used. Since there are so many robots in this large scale simulation, we omit explicit labeling of each robot (later implementation examples with smaller networks will show labels). In this figure, assigned robots are incident to edges in the network graph, while wanderers are not. Figure 6.7(a) shows the initial conditions. Since there is a wander within range of each network member, the leader-first-follower rule is applied, producing the network graph's first edge (see Figure 6.7(b)). While the first edge is being formed, there is only one robot assigned a formation position with a final field. Therefore, all the wanderers circle this position, waiting for vertex addition rules to be applicable. Once the leader-first-follower seed is formed, vertex addition rules are concurrently applied, as in Figure 6.7(c). Figures 6.7(d)-7(e) show further vertex additions being applied, as wanderers use the network's hop counters to navigate and satisfy the guards of the remaining vertex additions. Figure 6.7(f) shows the fully assembled formation.

6.7 Experimental Results

Here, we present experimental results from implementing the EGG system for formation assembly with a multi-robot network. First, we describe the network. Then, we present two experimental scenarios. In the first, the robots assemble a triangulation pattern. In the second, the robots assemble a line pattern.

6.7.1 The Multi-Robot Network

For the pre-Antarctic stages of this project, we use the prototype network shown in Figure 6.8. The prototype network is designed to approximate this network for experiments while the Antarctic platform is developed.

For mobility, each robot has a wheeled platform base. While these robots are not designed for snow and ice, the have controllable forward and angular velocity, dynamically similar to the tracked platform for the Antarctic. Each has an onboard computer. Communication between robots is achieved by 802.11g wireless communication modules on each robot. GPS receivers on each robot estimate the location of each robot, as well as heading information. Odometry sensors allow the robots to estimate their current positions relative

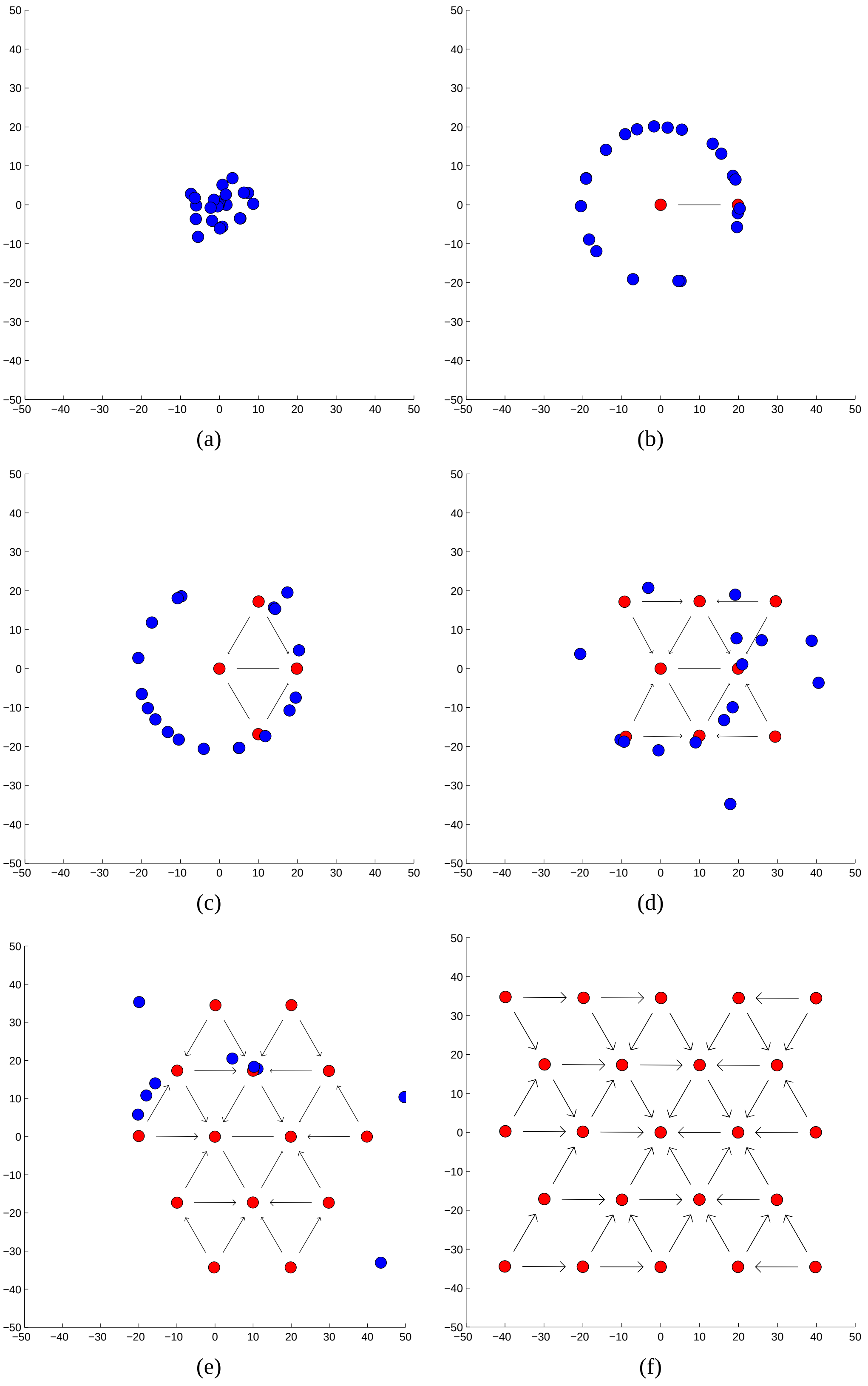


Figure 6.7. Triangulation scenario simulation. Assigned robots are incident to edges in the network graph, while wanderers are not. The robot achieves a desired formation that is much larger than the perception of any single robot.

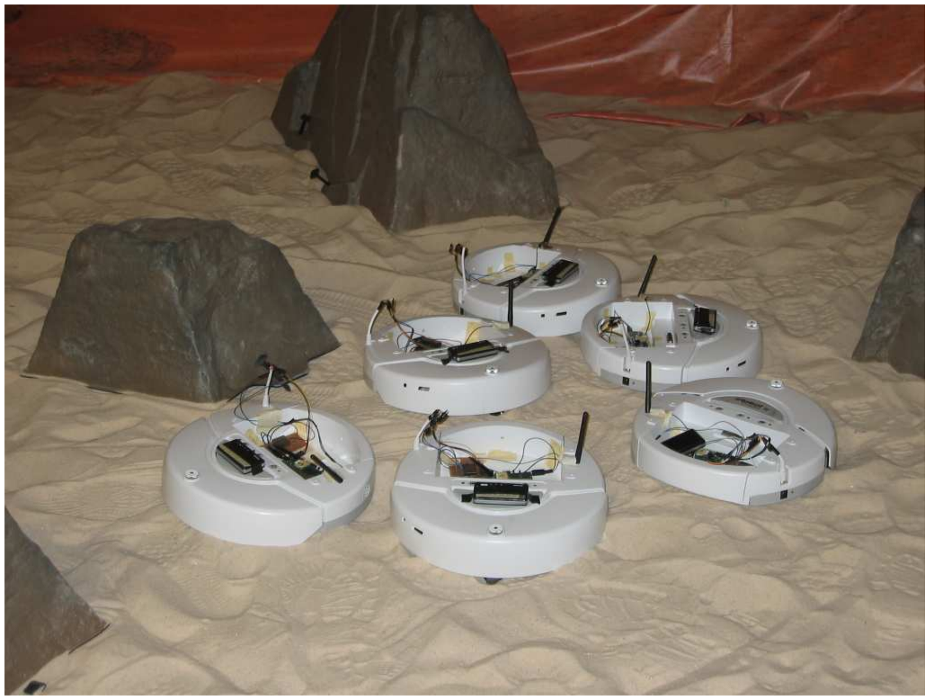


Figure 6.8. The multi-robot network. This network is used in the pre-Antarctic stages of this project.

to their initial positions.

Each robot has sensors for odometry information, represented as a pair of coordinates in $\mathbb{R}^2$, which we refer to as the robot's *odometry plane*. Initially, each robot assumes it is at the origin of its odometry plane with a heading corresponding to 0 rad from the odometry plane's *x*-axis. However, since each robot is initially at neither the same location nor necessarily the same heading, there is no agreement on the orientation of their odometry planes.

The wireless communication module provides each robot with the location of the other robots in the network. Since the network has global information, this allows for a great amount of freedom to limit what information is used by the robots. Since the actual sensor network will not have global information or global localization, we define a proximity range Δ, and allow the robots to only use the range and bearing information of robots within proximity range. In this manner, we can experiment with a variety of formations and proximity ranges. Also, by recording the GPS sensors during the experiments, we are able to verify and plot the results during each experiment. The EGG system for formation assembly only requires relative position estimation for the robots. As such, the method presented here for formation assembly can be utilized on multi-robot networks that can provide this information, regardless of their ability to localize.

6.7.2 Triangulation Implementation

To test our methods on an actual decentralized network, the previous scenario is implemented with $n = 6$ robots, as shown in Figure 6.9.

Since the network is small, we modify the wander control laws. Instead, each wanderer requests that the assigned robot with the lowest hop-counter in proximity range communicate the relative position of the next robot with the lowest hop-counter. In this manner, circular motion is avoided, and the wanderers directly move from one assigned robot to another, looking for applicable vertex addition rules.

In this scenario, each robot is given the EGG corresponding to the formation shown in Figure 6.1. Here, each constraint is 5 m long. The proximity range Δ is defined such that $\Delta = 6$ m.

In Figure 6.9, each robot is labeled with either w, indicating that it is a wanderer, or with its corresponding position in Figure 6.1, indicating that it is an assigned robot. In Figure 6.9(a), we see the initial setup, where each robot is a wanderer, labeled w. This implies that the leader-first-follower position rule is applicable, and it is applied. One of the robots is now a leader (labeled $\bar{p}_1$), and the other is a first-follower (labeled $\bar{p}_2$), and the follower begins moving to reach a distance of 5 m from the leader, as shown in Figure 6.9(b).

Note that, while vertex addition operations define a sequence of subgraphs, many vertex addition rules can be applied concurrently. As shown in Figures 6.9(c)-(d), two vertex addition position rules are applied simultaneously (robots labeled $\bar{p}_3$ and $\bar{p}_4$), before either has been finalized. This is because these rules depend only on the presence of two assigned vertices in $G(t)$, not on the entire subgraph before the corresponding vertex addition operation. In this way, this method takes advantage of concurrency to accomplish the formation

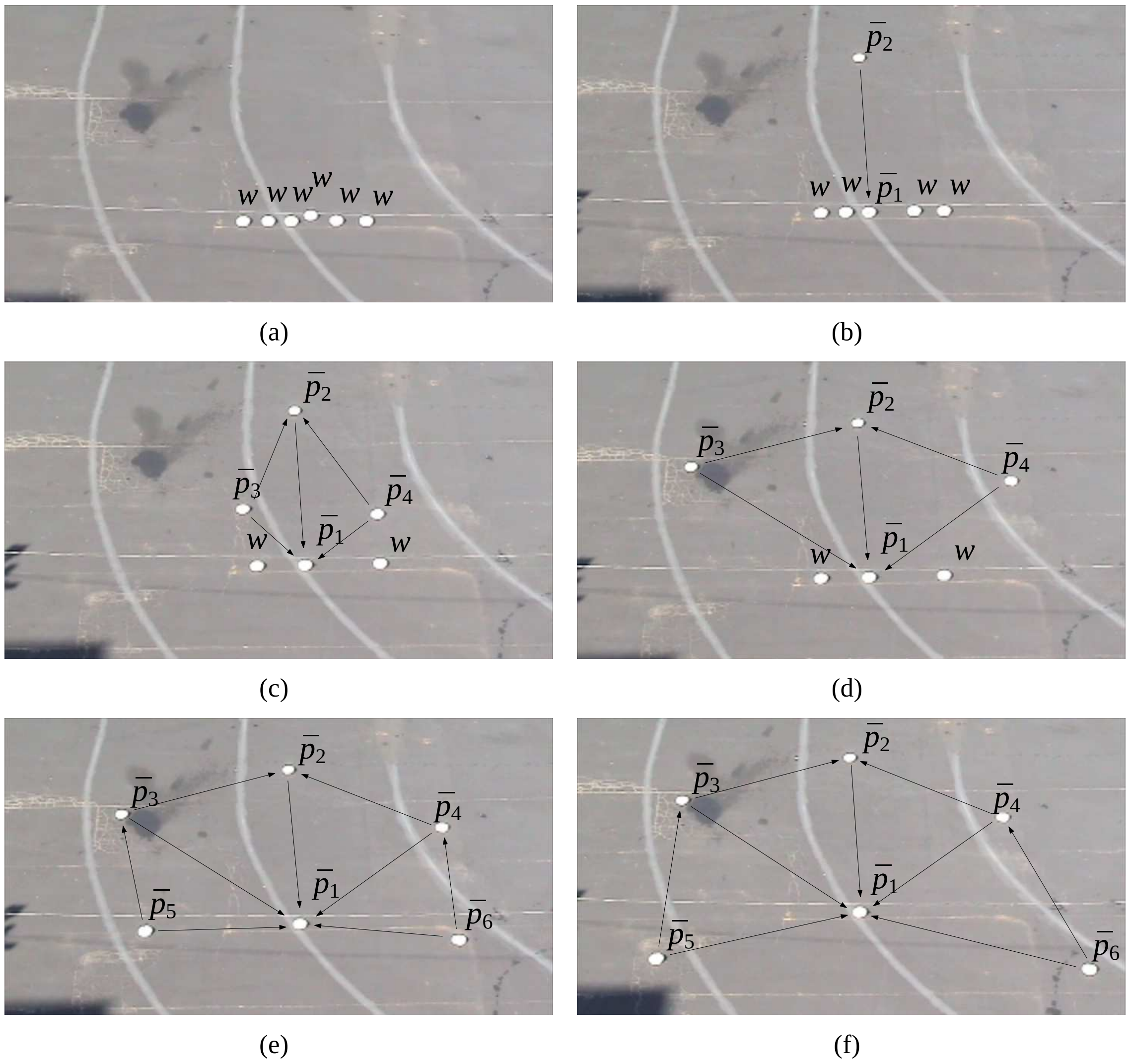
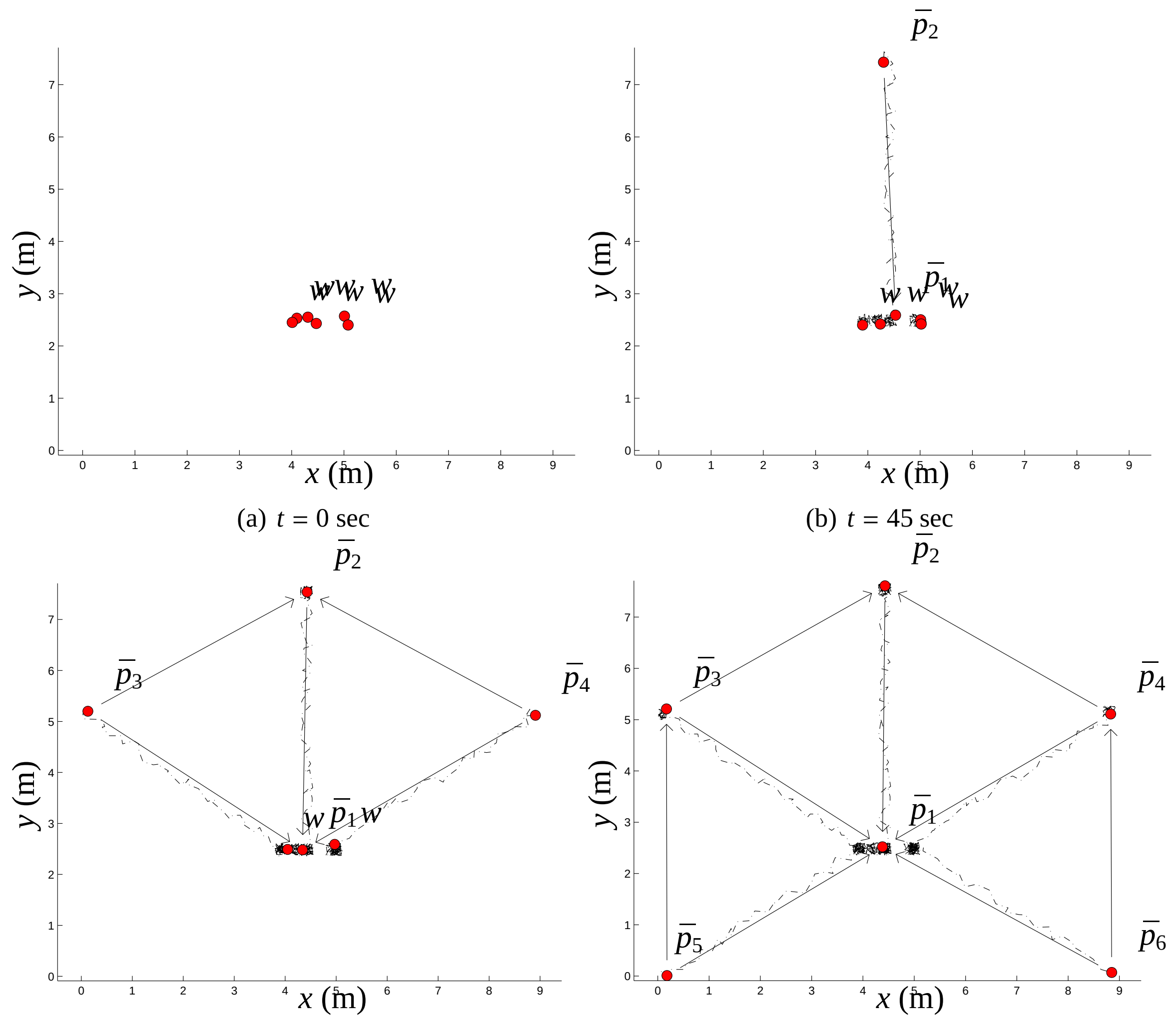


Figure 6.9. Triangulation scenario implemented on the multi-robot network.

task. Similarly, Figure 6.9(e) shows two concurrent vertex addition rules being applied. Finally, the EGG is successfully completed, as shown in Figure 6.9(f).

The error in the final formation (shown in Figure 6.9(f)) is due to the errors in the robots' GPS sensors, which have an average error of approximately 2 m. In Figure 6.10, the recorded GPS coordinates of the network during the assembly are plotted. Despite the noise and error in the GPS sensors, the robots accurately position themselves based on their estimations of the relative positions of other robots.

Figure 6.11 shows the constraint lengths in the network graph as they are established over time. Each constraint is established at approximately 5m as intended, despite the



(a) $t = 0$ sec

(b) $t = 45$ sec

(c) $t = 88$ sec

(d) $t = 120$ sec

Figure 6.10. The network trajectory during the triangulation assembly. This figure shows the GPS coordinates for each robot in the network.

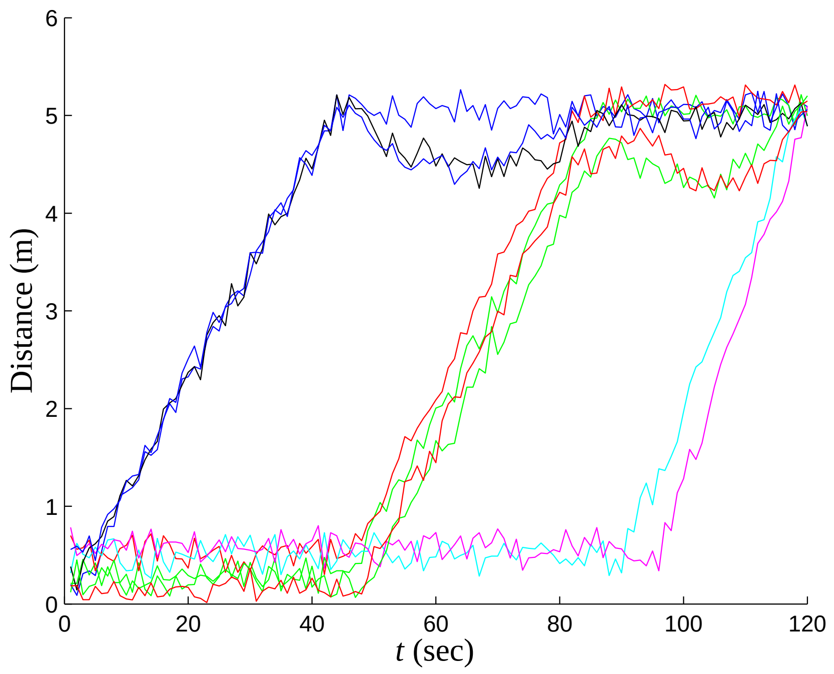


Figure 6.11. Network constraints during triangulation assembly.

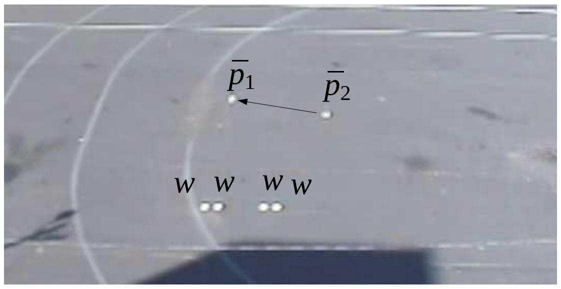
presence of noise and errors in the sensor data. Also, each constraint respects the proximity range $\Delta = 6$ m.

6.7.3 Line Implementation

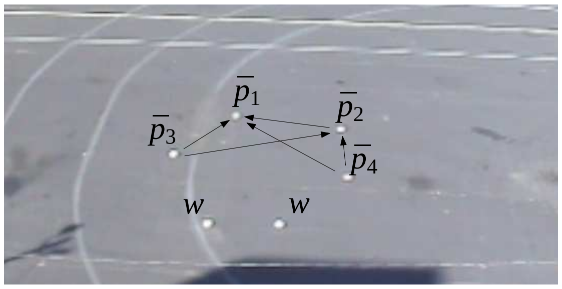
Here, we present a scenario for assembling a "straight line" formation of six robots. Each "link" in the line of robots represents a constraint 5 m. The resulting network graph has constraints of 5 m and 10 m. In this scenario, we use a proximity range Δ of 12 m. For the filming of this scenario, we initialize a leader-first-follower pair before the assembly process. Even with noise and error in the robots' sensors, as well as the collinearity in the desired formation, the formation is successfully assembled. Figure 6.12 shows the network performing the line formation.

In Figure 6.13, we show the GPS coordinates of the network as the line formation is assembled.

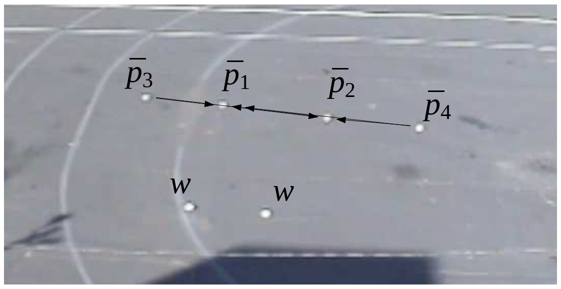
Figure 6.14 shows the 5 m and 10 m constraints being established in the network. Note that the network respects the proximity range $\Delta = 12$ m.



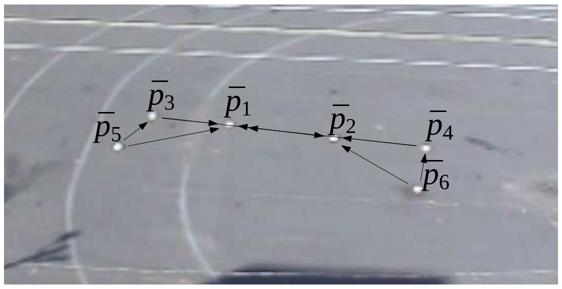
(a)



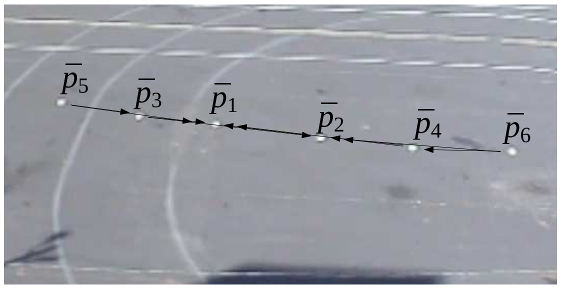
(b)



(c)

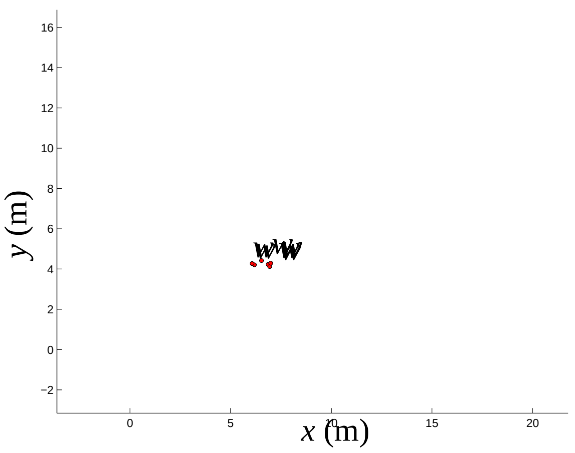


(d)

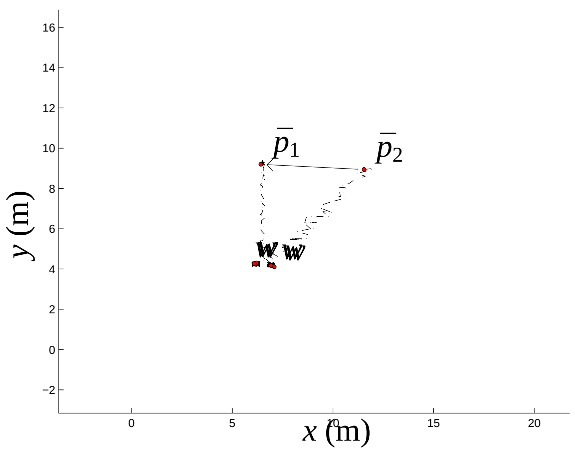


(e)

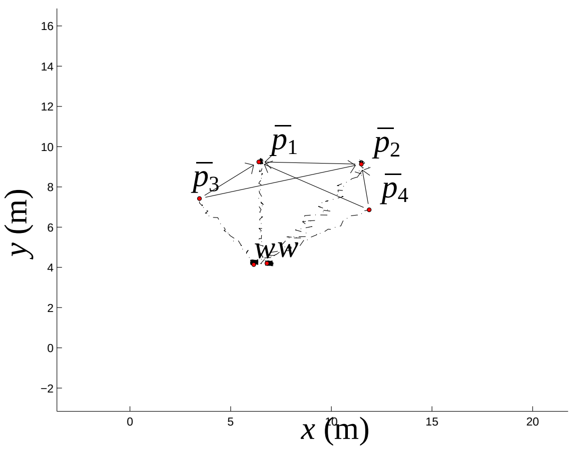
Figure 6.12. The robot network implementing a line formation.



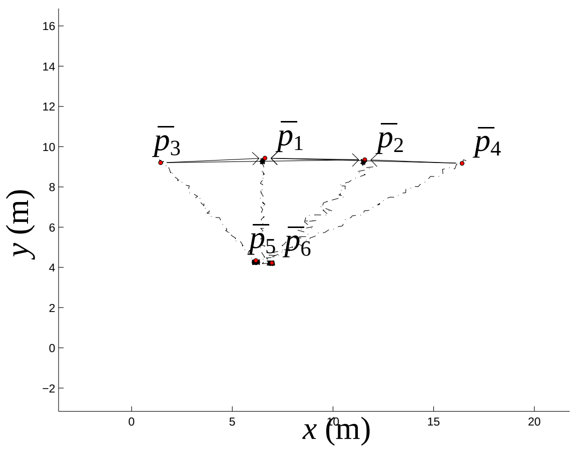
(a) $t = 0$ sec



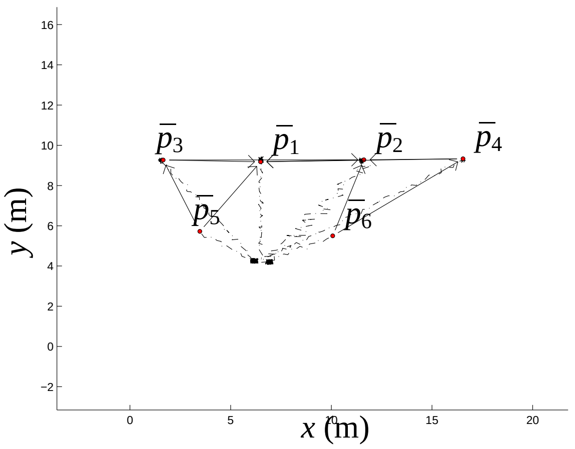
(b) $t = 110$ sec



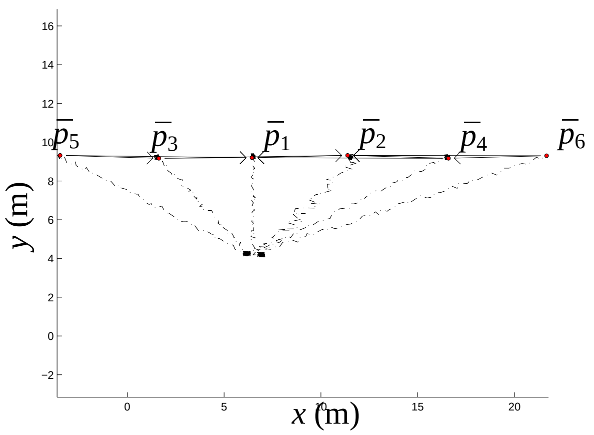
(c) $t = 150$ sec



(d) $t = 180$ sec



(e) $t = 200$ sec



(f) $t = 255$ sec

Figure 6.13. The network trajectory during the line assembly. This figure shows the GPS coordinates for each robot in the network.

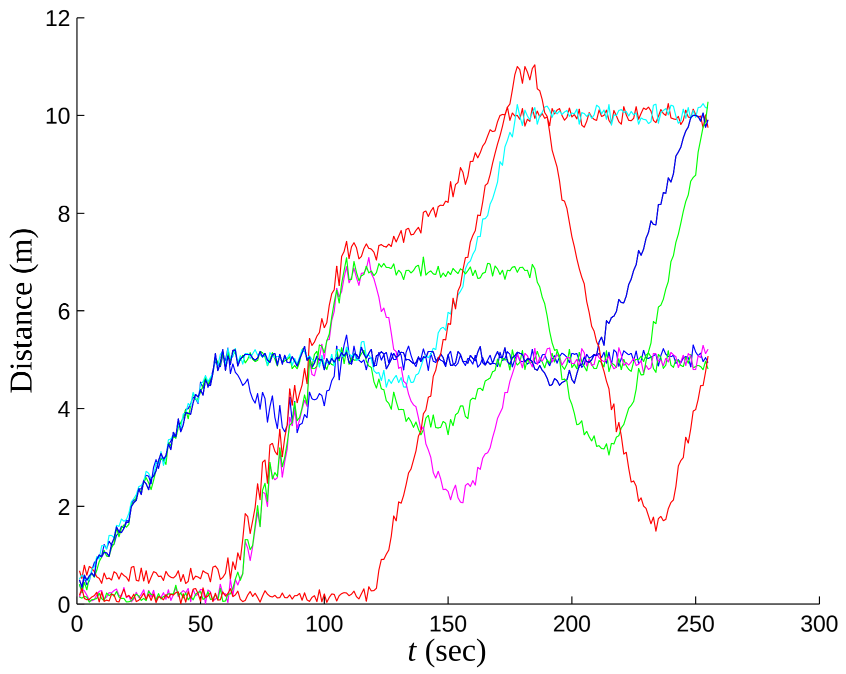


Figure 6.14. Network constraints during line assembly.

This EGG system for formation assembly will be used to automate multi-robot deployment, as presented in Chapter 7.

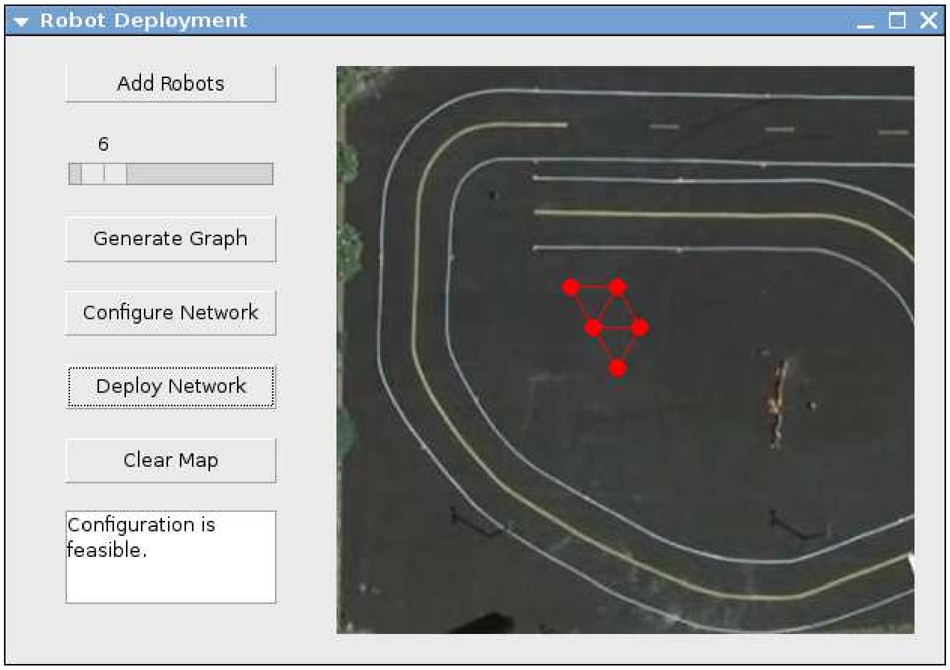
CHAPTER 7

DEPLOYMENT OF HETEROGENOUS MULTI-ROBOT NETWORKS WITH EMBEDDED GRAPH GRAMMAR SYSTEMS (EGGS)

One of the advantages of the methods presented in Chapters 5 and 6 is that the methods presented there can be implemented with multi-robot networks with little or no localization ability. In this chapter, we assume that the NASA scientists define a deployment for the network that must be satisfied in terms of the multi-robot network's location in the environment. We also assume that the multi-robot network has limited localization ability, in that some robots have localization ability, while others do not. As such, the network is *heterogeneous*, in that only certain robots can estimate their location in the environment. Such a lack of localization in a multi-robot network can arise out of design, for we show that localization ability is not necessary for all members of the network to achieve these goals. However, it can also arise out of failures, i.e., the failure of the localization systems of a subset of the network.

This chapter presents methods for automatically deploying the heterogeneous multi-robot network. Having a prototype multi-robot network composed of mobile robots shown in Figure 6.8, a user graphically enters a desired *network deployment* for the network. By "clicking" with a Graphical User Interface (GUI) incorporating satellite imagery shown in Figure 7.1, a network deployment is entered. This deployment represents the desired locations of the robots in the environment and the specific relative geometry that must be satisfied between pairs of robots. We present a method for automatically configuring the multi-robot network to deploy at the desired coordinates despite the limited localization ability of the robots.

This chapter concludes with experimental results that show the implementation of all of the methods in this work. Given a user-defined deployment and the network's proximity



Configuration is feasible.

Figure 7.1. Graphical User Interface (GUI) for configuring the network. A user enters the desired *deployment positions* in the GUI using satellite imagery. These positions correspond to the locations in the environment where we desire the robots to be located. Our methods automatically configure the network to implement the formation at the desired location.

range, the methods in Chapter 4 are used to define a stably persistent formation graph for the network. The EGG system for deployment presented in this chapter is defined from the desired deployment and this stable, persistent graph. This EGG is then given to each robot in the network. This EGG incorporates the EGG system for assembly presented in Chapter 6 which, when extended with the control laws and rules presented in this chapter, allow the network to navigate to the initial deployment location and assemble the desired formation at the location defined in the deployment. Then, the control laws from Chapter 5 are used to move the network while maintaining formation. In this manner, given a definition of the desired deployment and the network's proximity range, the multi-robot network satisfies the user-defined deployment automatically.

In Chapter 7.1, we characterize the limited localization ability of the network. Chapter 7.2 discusses control laws that allow robots to move to the same locations while preserving their connectivity. Chapter 7.3 presents an EGG system which allows robots without localization ability to form "chains", following robots with localization ability, so that the robots can navigate to the necessary location in the environment. We discuss how the robots transition from this new deployment EGG to a (slightly modified version of) the formation assembly EGG in Chapter 7.4. Finally, Chapter 7.5 presents a demonstration of these methods with an actual deployment scenario.

7.1 Preliminaries

Here, we review our definition of a network deployment. We also discuss the sensing limitations of the robots. Specifically, we discuss how only a proper subset of robots can estimate their location in the environment. Therefore, the problem is how to navigate the network to the location of the initial deployment despite this lack of localization for some of the network members.

7.1.1 Network Deployment

Recall from Chapter 2.2 we define desired network deployment with *deployment positions* $p_i : T \mapsto \mathbb{R}^2 \ \forall i \in N$ using the GUI shown in Figure 7.1. Each position corresponds to a location in the environment where we desire a robot to be located. Thus, these positions define a *network deployment* $P : T \mapsto \mathbb{R}^2$ such that $P(t) = \left[p_1(t)^T, \ldots, p_n(t)^T\right]^T$. For this problem, we assume there is no preference for specific robots to be assigned specific positions.

We assume that the deployment satisfies a formation such that, $\forall (i, j, t) \in N \times N \times T$, $\|p_i(t) - p_j(t)\| = \|p_i(0) - p_j(0)\|$. Therefore, the deployment defines a *formation* $\overline{P} \in \mathbb{R}^{2n}$ such that $\overline{P} = P(0)$. This also defines *formation positions* such that, $\forall i \in N$, $\overline{p_i} = p_i(0)$. In this case, the formation positions define both the relative geometry of the formation as well as the initial location of the network for the deployment.

7.1.2 Sensing Limitations: Proximity Range and Localization Abilities

All robots in the network have sensors for estimating the *relative positions* of robots within their *proximity range* $\Delta \in \mathbb{R}^+$. The proximity range is chosen to model the sensing limitations of the robots in the network. Hence, a pair of robots $(i, j) \in N \times N$ can sense and communicate with each other at time t if and only if $\|x_i(t) - x_j(t)\| \leq \Delta$. Therefore, this allows us to define $u_i(t)$ as a function of $x_i(t) - x_j(t)$ if and only if $\|x_i(t) - x_j(t)\| \leq \Delta$.

This network is heterogeneous in that not all robots have localization ability. We call a robot with localization ability a network *leader*. This localization ability implies that the leaders can estimate the relative position of the deployment positions. We define the indices of the network leaders as $N_l \subset N$, and insist that $|N_l| \geq 2$ (i.e., there must be at least one network leader). Given a specific location in the environment $p_i \in \mathbb{R}^2$, the localization ability of the leaders implies that, for all leader robots $i \in N_l$, we can define u_i as a function of $x_i - p_i$. The remaining robots in the network are *followers* who do not have localization ability. The followers cannot determine their relative positions to any defined goal location in the environment.

Since all robots do not have localization ability, we cannot simply assign robots to positions and have them navigate to each. In Chapter 6, we present a formation assembly EGG that respects the proximity range of the network. However, the formations are assembled using only the relative positions of the robots. Thus, the formation is assembled at the initial location of the network, and the formation is not achieved with any pre-defined, specific orientation relative to the environment.

The stable, persistent feasibility of the desired formation implies that the location and orientation of the formation is determined by the location of a specific pair of leader and first-follower robots in the persistent network graph. Therefore, our strategy is to assign leader robots to these positions in the formation. Then, once the followers have been led to the deployment positions, they can assemble the formation as presented in Chapter 6 using only the relative positions of other robots.

7.2 Network Deployment Control Laws

Here, we show how to navigate a pair of robots so they always stay within proximity range of each other, *preserving their connectivity*. Utilizing these control laws, a robot can follow another robot and navigate towards a common position in the environment. These control laws rely only on bounding the maximum velocities of the robots. This is a very reasonable assumption, since most systems have an inherent limit on their maximum velocities due to their design. These control laws are the foundation for our method of automatic network deployment.

7.2.1 Preserving Connectivity Between Pairs of Robots

The following theorem describes two robots in which one follows the other. By bounding the velocity of the robot being followed to a chosen maximum velocity, we guarantee that the pair of robots never lose connectivity.

Theorem 7.1 *Consider a pair of robots l and f. For a given proximity range $\Delta \in \mathbb{R}^+$ and maximum velocity $u_{max} \in \mathbb{R}^+$, robot f's control laws are defined by*

$$K = \frac{u_{max}}{\Delta}, \tag{23}$$

$$u_f = -K\left(x_f - x_l\right). \tag{24}$$

We assume the following about robot l:

- *The velocity of robot l is bounded by u_{max} such that $\|u_l(t)\| \leq u_{max}\ \forall t \in T$.*
- *$\bar{p_l} \in \mathbb{R}^2$ is a globally asymptotically stable equilibrium point of x_l.*

Then, for every initialization of the pair such that $\|x_f(0) - x_l(0)\| \leq \Delta$,

- *$\|x_f(t) - x_l(t)\| \leq \Delta\ \forall t \in T$, and*
- *$\bar{p_l}$ is a globally asymptotically stable equilibrium point of x_f.*

Proof: By our assumptions, x_l is Lipschitz continuous, since its first derivative is defined and its magnitude is bounded over the entire domain. From (24), x_f is continuously differentiable.

First, we show that connectivity is preserved. If $\|x_f(t) - x_l(t)\| \geq \Delta$, then this implies that

$$\begin{aligned}
\frac{d}{dt}\left(\frac{1}{2}\|x_f(t) - x_l(t)\|^2\right) &= \frac{1}{2}\frac{d}{dt}\left(\left(x_f(t) - x_l(t)\right)^T\left(x_f(t) - x_l(t)\right)\right) \\
&= \left(x_f(t) - x_l(t)\right)^T\left(\dot{x}_f(t) - \dot{x}_l(t)\right) \\
&= \left(x_f(t) - x_l(t)\right)^T\left(-K\left(x_f(t) - x_l(t)\right) - \dot{x}_l(t)\right) \\
&= -\frac{u_{max}}{\Delta}\|x_f(t) - x_l(t)\|^2 - \dot{x}_l(t)^T\left(x_f(t) - x_l(t)\right) \\
&\leq -u_{max}\|x_f(t) - x_l(t)\| + u_{max}\|x_f(t) - x_l(t)\| \\
&\leq 0.
\end{aligned} \tag{25}$$

This implies that the distance between the follower and the leader is always stable or decreasing if their distance is greater than or equal to the proximity range. If we assume for some $t \in T$ that $\|x_f(t) - x_l(t)\| > \Delta$, we always have a contradiction. If $t = 0$, then the initialization assumption is violated. If $t > 0$, then the continuous differentiability of x_f implies that, for some $t_\Delta \in (0, t]$, $\|x_f(t_\Delta) - x_l(t_\Delta)\| \geq \Delta$ and $\frac{d}{dt}\left(\frac{1}{2}\|x_f(t_\Delta) - x_l(t_\Delta)\|^2\right) > 0$. In other words, this implies that there is a time before t such that the distance between robots l and f is increasing while their distance is greater than or equal to the proximity range. This violates (25). Therefore, $\|x_f(t) - x_l(t)\| \leq \Delta \; \forall t \in T$.

To show that $\overline{p_l}$ is a globally asymptotically stable equilibrium point of x_f, we first define the translated system $\tilde{x}_f = x_f - \overline{p_l}$. The control laws in (23-24) imply that $\dot{\tilde{x}}_f = \dot{x}_f = -K(x_f - x_l)$. Similarly, we define the translated system $\tilde{x}_l = x_l - \overline{p_l}$. Substitution implies that

$$\dot{\tilde{x}}_f = -K\left(x_f - (\tilde{x}_l + \overline{p_l})\right) = -K(\tilde{x}_f - \tilde{x}_l).$$

Taken together, $\tilde{x}_f$ and $\tilde{x}_l$ are a cascade system. This implies that, if $\tilde{x}_l$ has a globally asymptotically stable origin, then so does $\tilde{x}_f$ [46]. Since we assume that $\tilde{x}_l$ has a globally asymptotically stable origin, then $\tilde{x}_f$ does as well. This implies that $\bar{p}_l$ is a globally asymptotically stable equilibrium point of x_f. ■

7.2.2 Navigating Leaders to Deployment Positions

To define the control laws of a leader robot driving to its assigned position for formation deployment, we implement the control law defined in the following theorem.

Theorem 7.2 *Consider a robot l. For a given proximity range $\Delta \in \mathbb{R}^+$, maximum velocity $u_{max} \in \mathbb{R}^+$, and formation position $\bar{p}_l \in \mathbb{R}^2$, we define the leader's control laws by*

$$K = \frac{u_{max}}{\Delta},$$
$$u_l(t) = \begin{cases} -u_{max}\frac{x_l(t)-p_l}{\|x_l(t)-p_l\|} & if\ \|x_l(t) - p_l\| > \Delta \\ -K\left(x_l(t) - \bar{p}_l\right) & if\ \|x_l(t) - \bar{p}_l\| \leq \Delta \end{cases} \tag{26}$$

This implies the following:

- *$\bar{p}_l$ is a globally asymptotically stable equilibrium point of x_l,*
- *x_l is continuously differentiable, and*
- *x_l is Lipschitz continuous.*

Proof: To show that p_l is a globally asymptotically equilibrium point of x_l, we define the translated system $\tilde{x}_l = x_l - \bar{p}_l$ and show that its origin is globally asymptotically stable. The Lyapunov function $V(\tilde{x}_l) = \frac{1}{2}\tilde{x}_l^T\tilde{x}_l$ is globally positive definite. Its time derivative is

$$\dot{V}(\tilde{x}_l) = \tilde{x}_l^T\dot{\tilde{x}}_l = \begin{cases} -u_{max}\|\tilde{x}_l\| & \text{if } \tilde{x}_l > \Delta \\ -K\|\tilde{x}_l\|^2 & \text{if } \tilde{x}_l \leq \Delta \end{cases}.$$

Therefore, $\dot{V}(\tilde{x}_l)$ is globally negative definite. Since the same Lyapunov function applies when $\tilde{x}_l > \Delta$ and $\tilde{x}_l \leq \Delta$, this implies that the origin of $\tilde{x}_l$ is globally asymptotically stable. This implies that $\bar{p}_l$ is a globally asymptotically stable equilibrium point of x_l.

If the leader is initialized outside of proximity range of $\bar{p}_l$ such that $\|x_l(0) - p_l\| > \Delta$, note that the control laws in (26) define $\dot{x}_l(0)$ to be a velocity with a magnitude of u_{max} in the direction of $\bar{p}_l$. This velocity will be constant until time t_Δ, when $\|x_l(t_\Delta) - \bar{p}_l\| = \Delta$. This implies that $x_l(t)$ is continuously differentiable over $(0, t_\Delta)$. Note that the limit as $t \to t_\Delta$ from the left is

$$\lim_{t \to t_\Delta^-} \dot{x}_l(t) = -u_{max} \frac{x_l(t_\Delta) - \bar{p}_l}{\|x_l(t_\Delta) - \bar{p}_l\|}.$$

At t_Δ, $\|x_l(t_\Delta) - \bar{p}_l\| = \Delta$ the control laws switch such that $\dot{x}_l(t_\Delta) = -K\left(x_l(t_\Delta) - \bar{p}_l\right)$. Hence, $x_l(t)$ is continuously differentiable over (t_Δ, ∞). Also, the limit as $t \to t_\Delta$ from the right is

$$\begin{aligned}\lim_{t \to t_\Delta^+} \dot{x}_l(t) &= -K\left(x_l(t_\Delta) - \bar{p}_l\right) \\ &= -\frac{u_{max}}{\Delta} \frac{x_l(t_\Delta) - \bar{p}_l}{\|x_l(t_\Delta) - \bar{p}_l\|} \|x_l(t_\Delta) - \bar{p}_l\| \\ &= -u_{max} \frac{x_l(t_\Delta) - \bar{p}_l}{\|x_l(t_\Delta) - \bar{p}_l\|} = \lim_{t \to t_\Delta^-} \dot{x}_l(t).\end{aligned}$$

Since the limit as $t \to t_\Delta$ is identical from the left and from the right, then $\dot{x}_l$ is continuous over the entire domain, and x_l is continuously differentiable. Since $\|\dot{x}_l(t)\| \le u_{max}\ \forall t \in T$, this implies that x_l is Lipschitz continuous. ∎

The following corollary shows that, when following another robot with a constant, bounded velocity, the relative position of the leading and following robot stabilizes to a position Δ apart from each other, with the follower directly "behind" the leader.

Corollary 7.3 *Consider again the pair of robots l and f, with proximity range $\Delta \in \mathbb{R}^+$. Define $\hat{u} \in \mathbb{R}^2$ as a constant unit vector. Assume that robot l has a constant velocity in the direction of $\hat{u}$ with a magnitude of $u_{max} \in \mathbb{R}^+$ such that, $\forall t \in T$, $\dot{x}_l(t) = u_{max}\hat{u}$. Assume that u_f is defined as in Theorem 7.1, and that $\|x_l(0) - x_f(0)\| \le \Delta$. This implies that $-\Delta\hat{u}$ is a globally asymptotically stable equilibrium point of $\tilde{x}_f = x_f - x_l$.*

Proof: We define $\hat{x}_f = x_f - x_l + \Delta\hat{u}$. This implies that

$$\dot{\hat{x}}_f = \dot{x}_f - \dot{x}_l = -K(x_f - x_l) - u_{max}\hat{u} = -K(x_f - x_l + \Delta\hat{u}) = -K\hat{x}_f.$$

The system $\dot{\hat{x}}_f = -K\hat{x}_f$ has a globally, exponentially stable origin. This implies that $-\Delta\hat{u}$ is a globally, exponentially stable equilibrium point of $\tilde{x}_f$. ■

Corollary 7.3 implies that we should choose a "safe" proximity range for the network. Ideally, the chosen proximity range should be well enough within the actual limits of the robot sensors to allow for the noise and potential error of the system.

Theorem 7.1 shows us that, as long as the velocity of the leader robot is defined and bounded by our chosen maximum velocity, the distance between the pair of robots cannot exceed the proximity range. A corollary of Theorem 7.1 is that the state and dynamics of robot f also satisfy the same assumptions made about robot l.

Corollary 7.4 *Consider the pair of robots l and f. Given the same assumptions and initialization as in Theorem 7.1, then the velocity of the follower robot is bounded by u_{max} such that $\|u_f(t)\| \le u_{max}\ \forall t \in T$.*

Proof: The control law for robot f in (23-24) implies that

$$\|u_f(t)\| = |-K|\|x_f(t) - x_l(t)\| = \frac{u_{max}}{\Delta}\|x_f(t) - x_l(t)\|.$$

Since $\|x_f(t) - x_l(t)\| \le \Delta\ \forall t \in T$, then

$$\|u_f(t)\| \le \frac{u_{max}\Delta}{\Delta} = u_{max}.$$

Hence, $\|u_f(t)\| \le u_{max}\ \forall t \in T$. ■

Corollary 7.4 implies that, while robot f is following robot l, another robot within proximity range of robot f could follow robot f in the same manner and never loose connectivity with robot f. This suggests that we can connect robots together to follower a single leader using these control laws in Theorems 7.1 and 7.2.

7.3 An Embedded Graph Grammar (EGG) System for Deployment

Here, we present a method to automatically generate *Embedded Graph Grammar (EGG) systems* that allows follower robots (without localization) to follow leader robots (with localization). Thus, the network can navigate to a common goal in the environment. The EGG will use the control laws from Chapter 7.2. With this EGG, the entire network can navigate to a common location, and it requires only one robot to have localization (i.e., it requires only one leader robot). Later we describe how two robots with localization allow the network to implement this EGG system in conjunction with the assembly EGG system from Chapter 6 to assemble the formation at the initial deployment location.

7.3.1 Embedded Graph Grammar System for Deployment

For the EGG we present here, we define the label set of each vertex such that each label has two parts: the *mode* and the *go flag*. The mode indicates what control law the robot is implementing, while the go flag is a boolean indicating whether or not the robot can implement the control law. If the go flag is $false$, the robot must sets its control to zero and stay at the same location. We use the notation that $l(v_i).mode$ is the mode of robot i, and $l(v_i).go$ is the go flag of robot i.

Since this is a heterogeneous network, the leaders and the followers will each have different modes. We define mode L as *leader mode*. When in leader mode, the robot's control law is designed to stabilize the robot to a given goal position for deployment. Thus, $l(v_i).mode = L$ and $l(v_i).go = true$ if and only if $i \in N_l$ and robot i is moving towards its assigned deployment location.

The follower robots initially are assigned mode U, which is *unassigned mode*. This mode corresponds to a follower that has no one to follow, and does not move. Once it has been assigned someone to follow, it changes its mode to F, which is *assigned follower mode*. Robots in assigned follower mode have a single edge in the network graph directed towards the robot they are following. Thus, $l(v_i).mode = F$ and $l(v_i).go = true$ indicates that robot i is following a leader. If $l(v_i).go = false$, the robot does not move.

7.3.2 Initial Network Graph

To describe the required initial conditions of this EGG system, we use a *proximity graph* $\mathbb{G}(t) = (\mathbb{V}, \mathbb{E}(t))$, as presented in Chapter 2.3. Here, $\mathbb{V} = V$, the vertices in our network graph, and $\exists(v_i, v_j) \in \mathbb{E}(t)$ if and only if $v_i \neq v_j$ and $\|x_i(t) - x_j(t)\| \leq \Delta$, and we say that robots i and j are *connected*. Hence, edges in our proximity graph indicate which pairs of robots can sense and communicate with each other.

We initialize the leader robots in mode L, the follower robots in mode U, and the go flags of all robots are set to $false$. Thus, at $t = 0$, the network graph $G(0)$ has no edges. We initially require that a path exists in the proximity graph $\mathbb{G}(0)$ between each follower and at least one leader. As long as this condition is satisfied, the initial proximity graph can be disconnected. The following describes the EGG rules for "linking" pairs of robots.

7.3.3 Linking Robots with Linking Rules

To establish constraints between leaders and followers, we first use *linking rules* shown in Figure 7.2. There are two linking rules. In the first linking rule, the left graph consists of two robots. One is a leader robot, and one is an unassigned follower. The left graph has no edges. If they are within proximity range, the follower can switch to follow mode and add an edge to the network graph directed towards the leader. The other linking rule is identical except that, instead of a leader robot, the left graph includes an assigned follower. By repeatedly applying this rule, all unassigned followers are assigned to follow either a leader or a predecessor of a leader.

7.3.4 Moving Robots with Go Rules

The *go rules* specify when the leaders and assigned followers can begin implementing their assigned control laws, as shown in Figure 7.3. For either a leader or an assigned follower, if all its immediate predecessors have *true* go flags and if no robots within proximity range to it are unassigned followers, the robot switches its go flag from $false$ to *true* and begins implementing its control laws. These rules guarantee that all unassigned followers are

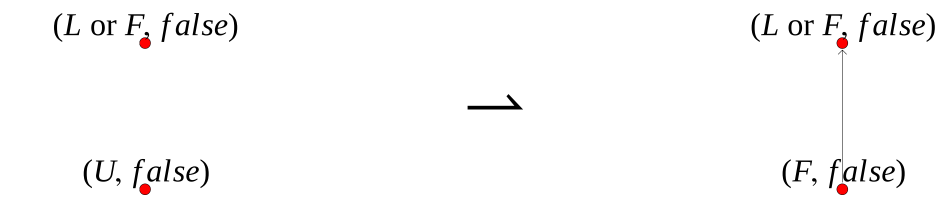


Figure 7.2. Linking rules. An unassigned follower will assign itself to follow a leader or an assigned follower. Here, the left part of the figure indicates the left graph of the rule, while the right part represents the right graph. This figure represents two rules, one for assigning to follow a leader, and one for assigning to follow an assigned follower. This rule does not change the label of the robot being followed. The guard requires that both robots are within proximity range of each other.

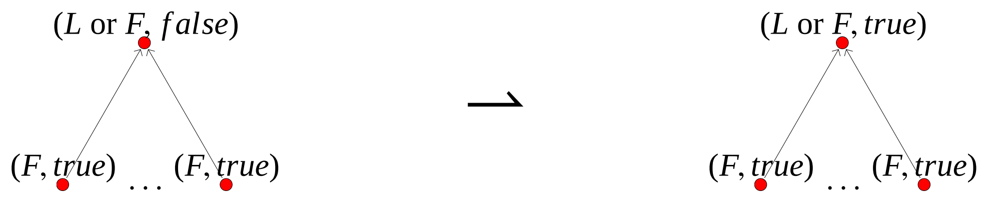


Figure 7.3. Go rules. If, for a leader or assigned follower with a $false$ **go flag, all its immediate predecessors have a** *true* **go flag, and no unassigned followers are in proximity range, it can switch its go flag to true.**

linked to follow a leader or a predecessor of a leader before the robots within their proximity range move.

7.3.5 Break Rules

While the previous rules ensure that robots become predecessors of leaders, and that robots do not leave unassigned followers behind, it is possible for multiple followers to follow the same robot. If that robot has a constant velocity, then Corollary 7.3 implies that the followers will stabilize to the same location, directly behind the leading robot. For our multi-robot system, this is not desirable, since the robots need to avoid colliding. Therefore, we define *break rules* that reduce the immediate predecessors of a single robot. The left graph has a robot being followed by two follower robots. If all of these robots are within proximity range of each other, one of the following robots is switched to follow the other follower. For any robot with multiple immediate predecessors, the repeated application of

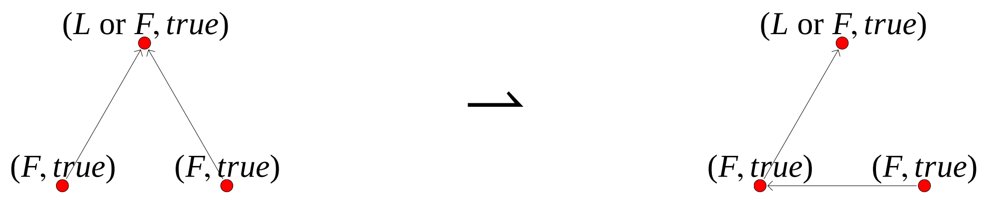


Figure 7.4. Break rules. This rule prevents more than one robot from following the same robot at the same and stabilizing to the same position. If two followers are following the same robot, one of the followers will switch to follower the other follower. The guard function requires all robots to be within proximity range of each other.

this rule ensures that, eventually, it will only have one immediate predecessor. Figure 7.4 represents these break rules.

When implemented with our multi-robot network, this EGG results in "chains" of robots, as shown in Figure 7.7. In general, this EGG system for deployment can be used with only one network leader to allow any number of followers to navigate to a desired location. Next, we describe how the desired formation is assembled such that these followers are assigned and navigate towards unique deployment positions, despite their lack of localization ability.

7.4 Formation Assembly

Here, we discuss how to assemble formations as the network arrives at the deployment coordinates in the environment. Recall from Chapter 6 our methods to automatically generate an EGG system for assembling persistently feasible formations. This assembly strategy is based on EGG rules that correspond to *graph operations*. In this previous work, the network graph is initially unassembled, and each robot begins in a *wander* mode. An initial edge is added between a *leader* and *first-follower* pair of robots. The positions of these robots specify the location and orientation of the formation in the environment. Then, *vertex addition rules* attach wanderers to the leader and first-follower, as shown in Figure 7.5. These vertex addition rules assign the robots their unique positions. These robots then navigate to the correct locations using only the *relative positions* of other robots. When all

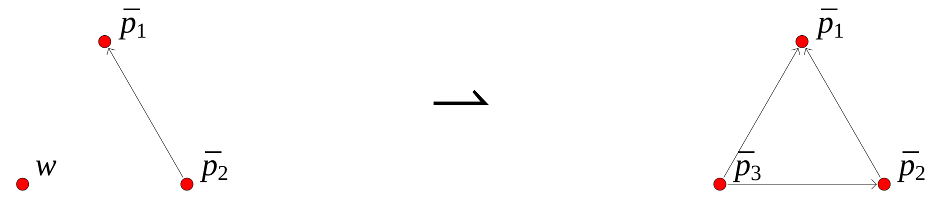


Figure 7.5. Vertex addition rules.

vertex addition rules have been applied, the formation is assembled.

The initial conditions of the EGG presented in Chapter 6.2 are that all wander robots must be within proximity range of the leader robot in the formation. While our EGG for linking robots allows us to navigate the network with only one leader, we utilize two leader robots in our current implementation. In this implementation, we order the deployment positions such that $\bar{p}_1$ is the *leader position* and $\bar{p}_2$ is the *first-follower position*. Both leaders are initially assigned to navigate to $\bar{p}_1$. When a leader has successfully navigated to this position, it assigns its mode to $\bar{p}_1$ such that $l(v_i).mode = \bar{p}_1$. The remaining leader, when encountering the robot assigned to $\bar{p}_1$ will then switch and begin converging to the first-follower position $\bar{p}_2$. Therefore, the network deploys as two chains, one which converges to the leader position in the formation graph, and one that converges to the first-follower position.

In order to switch followers from follow mode to wander mode, we employ *wander rules* that assign followers to wander mode once they have reached the deployment location. If a follower is following a leader robot, and all its immediate predecessors are within proximity range of that leader, it switches to wander mode, and all its predecessors begin following the leader or first-follower. Figure 7.6 depicts wander rules. The application of this rule implies that all wanderers are within proximity range of either the leader or the first-follower, which are sufficient conditions for successfully assembling the formation.

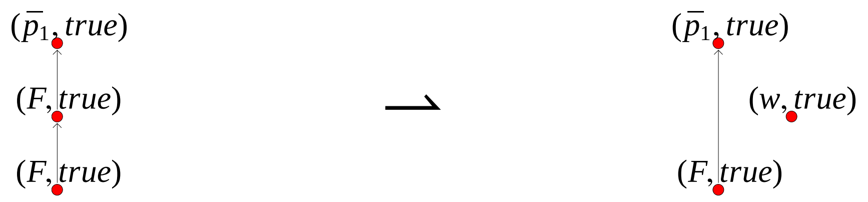


Figure 7.6. Wander rules. If a follower is following a robot that is assigned a position in the formation, and all of its predecessors are within proximity range of the robot assigned a formation position, it can break off from the chain and become a wanderer. All of its predecessors begin following the robot assigned a formation position.

7.5 Implementation

Here, we discuss the implementation results of the automatic EGG generated for formation deployment. This demonstration utilizes all of the methods we present in this work.

We implement the deployment depicted in Figure 7.1, where a user has chosen $n = 5$ deployment positions using the GUI with satellite imagery of our test field. The GUI is able to compare the positions with the known coordinates of reference positions in the satellite image to estimate the desired deployment positions for the network. Using the methods in Chapter 4 and the defined proximity range $\Delta = 6$ m, the software determines that the deployment positions can be assembled as a persistent formation. A stable, persistent graph is generated for the formation, and the network members are automatically configured to assemble the formation using the methods in Chapter 6. All network members are configured to implement the EGG system for linking robots discussed in Chapter 7.3. Then the network members are given the command to deploy.

Figure 7.7 shows the network deploy and assemble the initial formation. Figure 7.7(a) shows the initial state of the network. The leaders are in leader mode, labeled L, and the followers are initially in unassigned mode, labeled U. The robots begin applying the EGG rules described in Chapter 7.3. The two leader robots initially navigate to the leader position $\bar{p}_1$. However, since they are in range of each other, one of the leader robots switches to navigate to the first-follower position $\bar{p}_2$. By applying linking rules, two followers begin following the leader on the right, and one follower begins following the leader on the right. Since two follower are following the right leader, the breaking rule is applicable, and is applied, allowing the followers to follow as a "chain", seen in Figure 7.7(b).

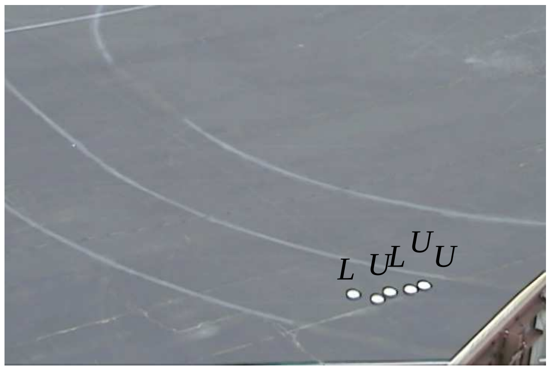
As seen in Figure 7.7(c), the leader on the left arrives at formation position $\bar{p}_1$, the leader position for the formation. It changes to the appropriate mode, allowing its follower to switch to wander mode. Figure 7.7(d) shows this wander perform a vertex addition, assigning itself formation position $\bar{p}_3$. Also, the first-follower arrives at $\bar{p}_2$, and one of its following robots "pops off" as a wanderer. The follower who previously followed the wanderer now follows the first-follower (assigned $\bar{p}_2$). Figure 7.7(e) shows the wanderer performing a vertex operation, assigning itself $\bar{p}_4$. Also, the last follower becomes a wanderer. While these vertex addition operations are being completed, the wanderer is already within proximity range of the leader, which has an available vertex addition operation. Thus, it does not move until the vertex addition for $\bar{p}_3$ is complete. Once these vertex operations are complete (Figure 7.7(f)), the remaining wanderer navigates through the network using the hop counters, and finally performs the last vertex operation (Figure 7.7(g)), assigning itself $\bar{p}_5$. The entire formation is assembled, as shown in Figure 7.7(h).

Figure 7.8 shows the GPS coordinates of the robots during the formation deployment. The robots successfully deploy the desired formation.

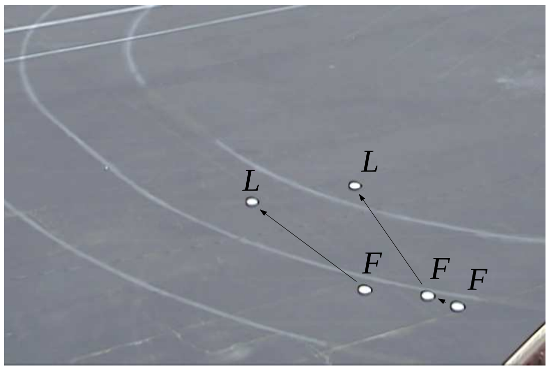
Figure 7.9 shows how the followers respected the proximity range as edges between leaders and followers are created and destroyed. Note that all edges always have a length less than the proximity range $\Delta = 6$ m.

Figure 7.10 shows the edges in the network graph that correspond to the formation. All edges stabilize to a desired length of 5 m. Also, these edges never violate the proximity range of $\Delta = 6$ m.

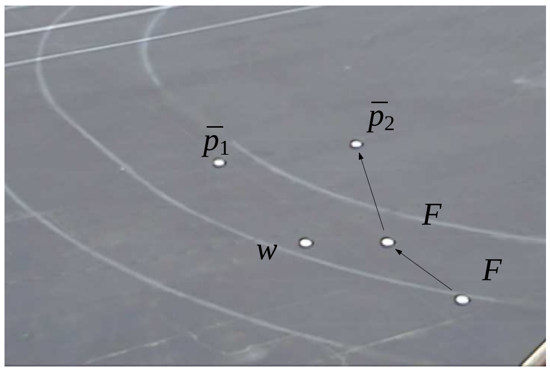
Once assembled, the network begins following the deployment, as shown in Figure 7.11. The leader begins driving "north", as shown in Figure 7.11(a). The rest of the network members adjust their velocities using the control laws from Chapter 5, and the entire



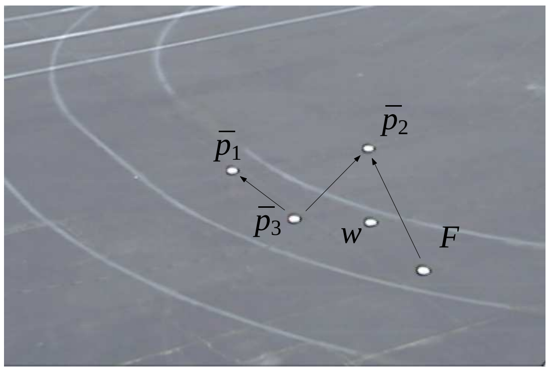
(a)



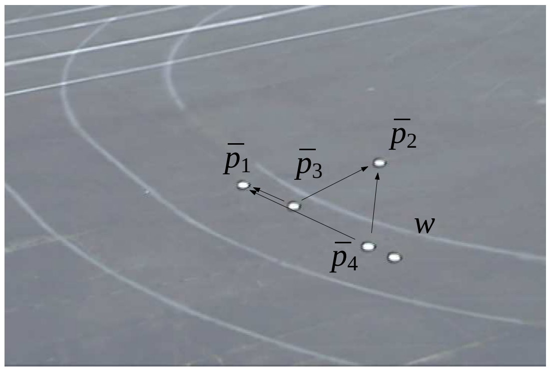
(b)



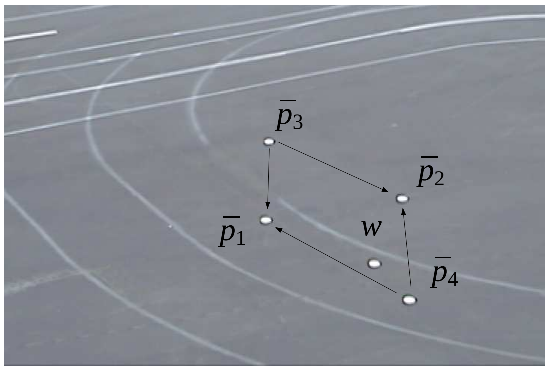
(c)



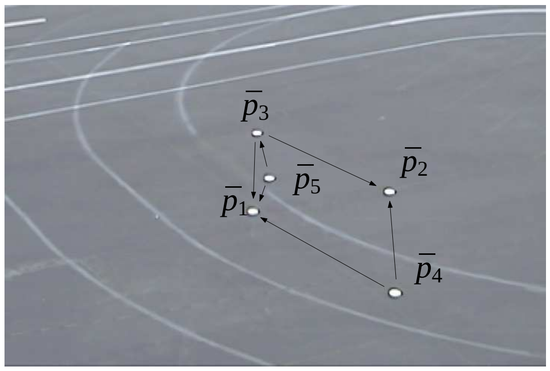
(d)



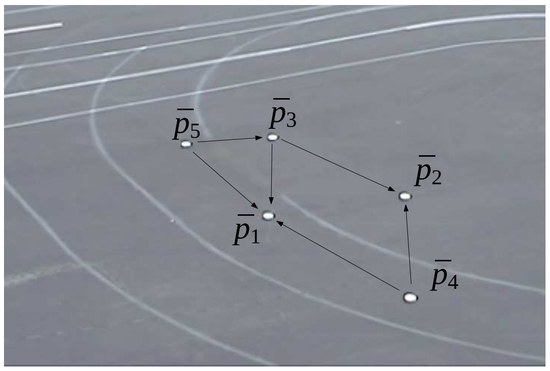
(e)



(f)

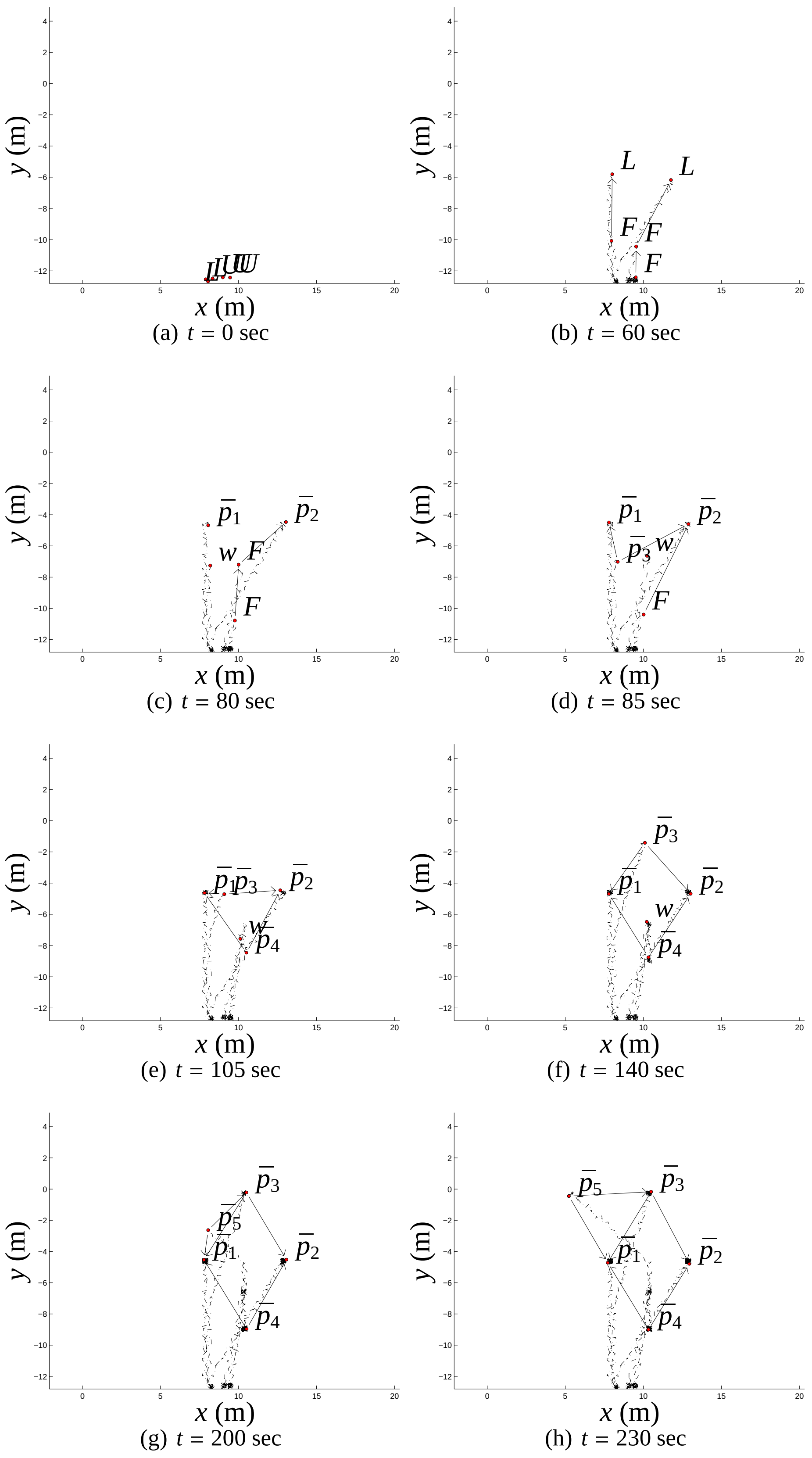


(g)



(h)

Figure 7.7. Heterogenous multi-robot deployment. The robots use the EGG system presented here to deploy to the initial location of the deployment and assemble the formation.



(a) $t = 0$ sec

(b) $t = 60$ sec

(c) $t = 80$ sec

(d) $t = 85$ sec

(e) $t = 105$ sec

(f) $t = 140$ sec

(g) $t = 200$ sec

(h) $t = 230$ sec

Figure 7.8. Network trajectory during deployment. This data is taken from GPS logs of the robots and shows their perspective of the network. The robots successfully deploy the desired formation at the correct location in the environment.

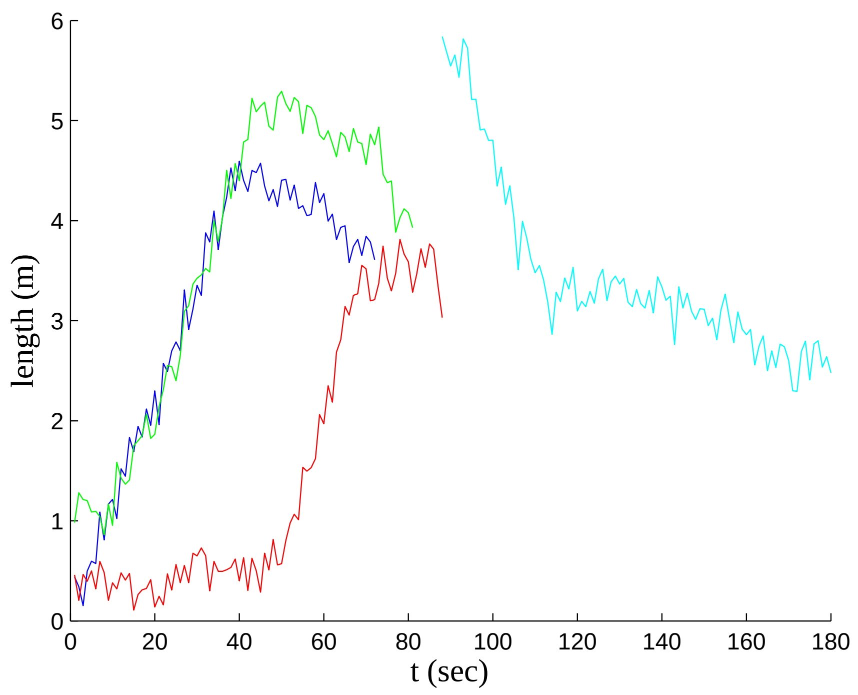


Figure 7.9. Follower edges. These plots show the edge length of the edges between followers and leaders during the deployment. All edges have a length less than $\Delta = 6$ m.

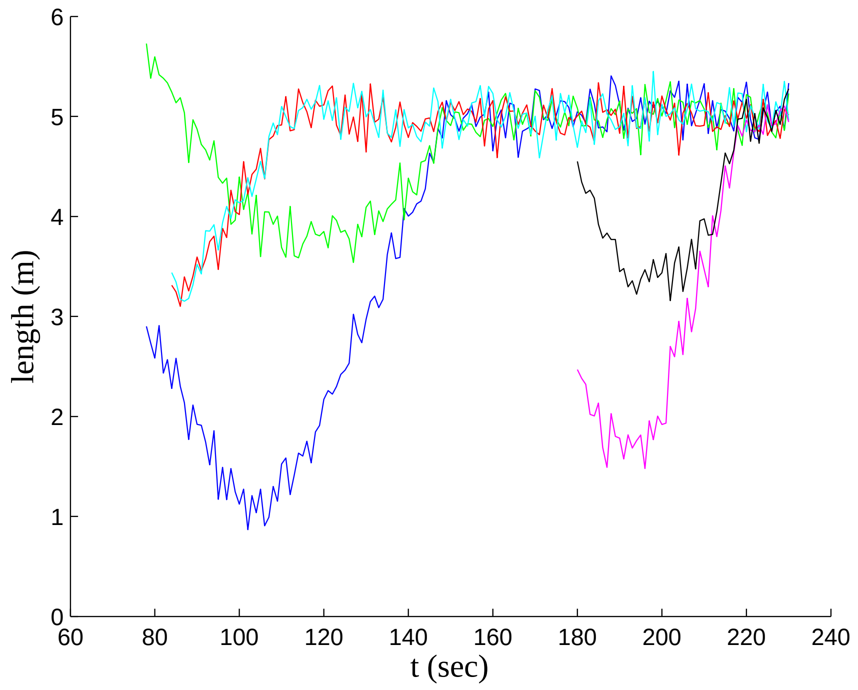


Figure 7.10. Formation network graph edges. These are the edges for the assembled formation. Since these edges correspond to an equilateral triangulation, all edges converge to a length of 5 m as the formation is assembled. These edges do not exceed the proximity range $\Delta = 6$ m.

formation moves north. Eventually, the leader reaches the end and stops, as shown in Figure 7.11(d). The rest of the network stabilizes to this new position for the formation, seen in Figure 7.11(f).

Figure 7.12 shows the GPS coordinates of the robots during the formation motion. The robots maintain formation, and stabilize to the desired formation at the final location for the deployment.

Figure 7.13 shows the network graph edge lengths during the formation motion. Initially, the edges are approximately 5 m long. During the formation motion, error is introduced as the robots attempt to estimate their appropriate velocities. Note that the proximity range is violated by two “peaks” in the error. The largest peak violates the proximity range by approximately .1 m. This reiterates the necessity of the proximity range to be well within the operating range of the network, to account for noise and errors. When the formation stops moving, the edges stabilize back to lengths of 5 m.

This experiment demonstrates all of the methods detailed in this work.

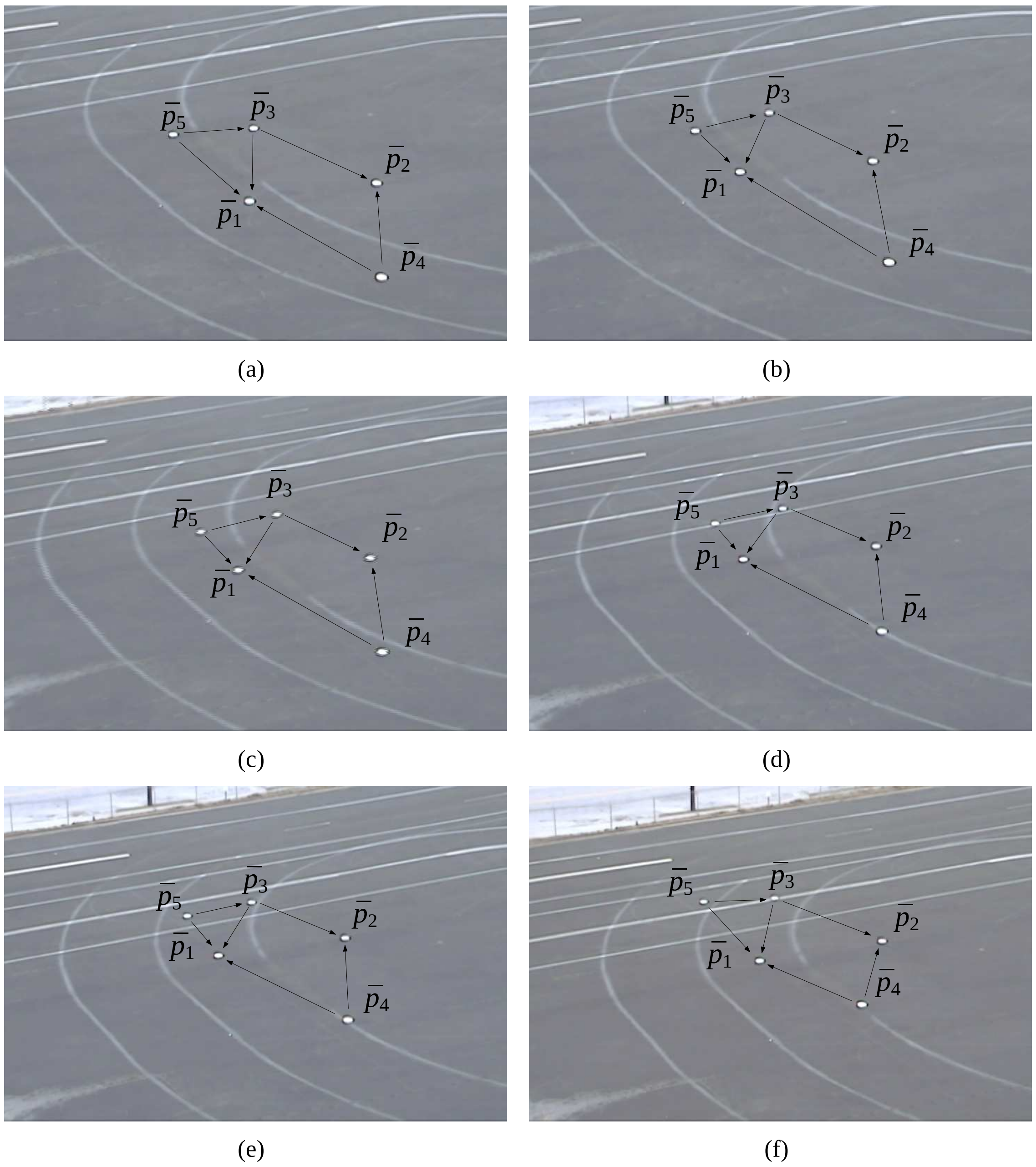


Figure 7.11. Moving in formation. The robots employ the control laws in Chapter 5 to maintain the formation during motion.

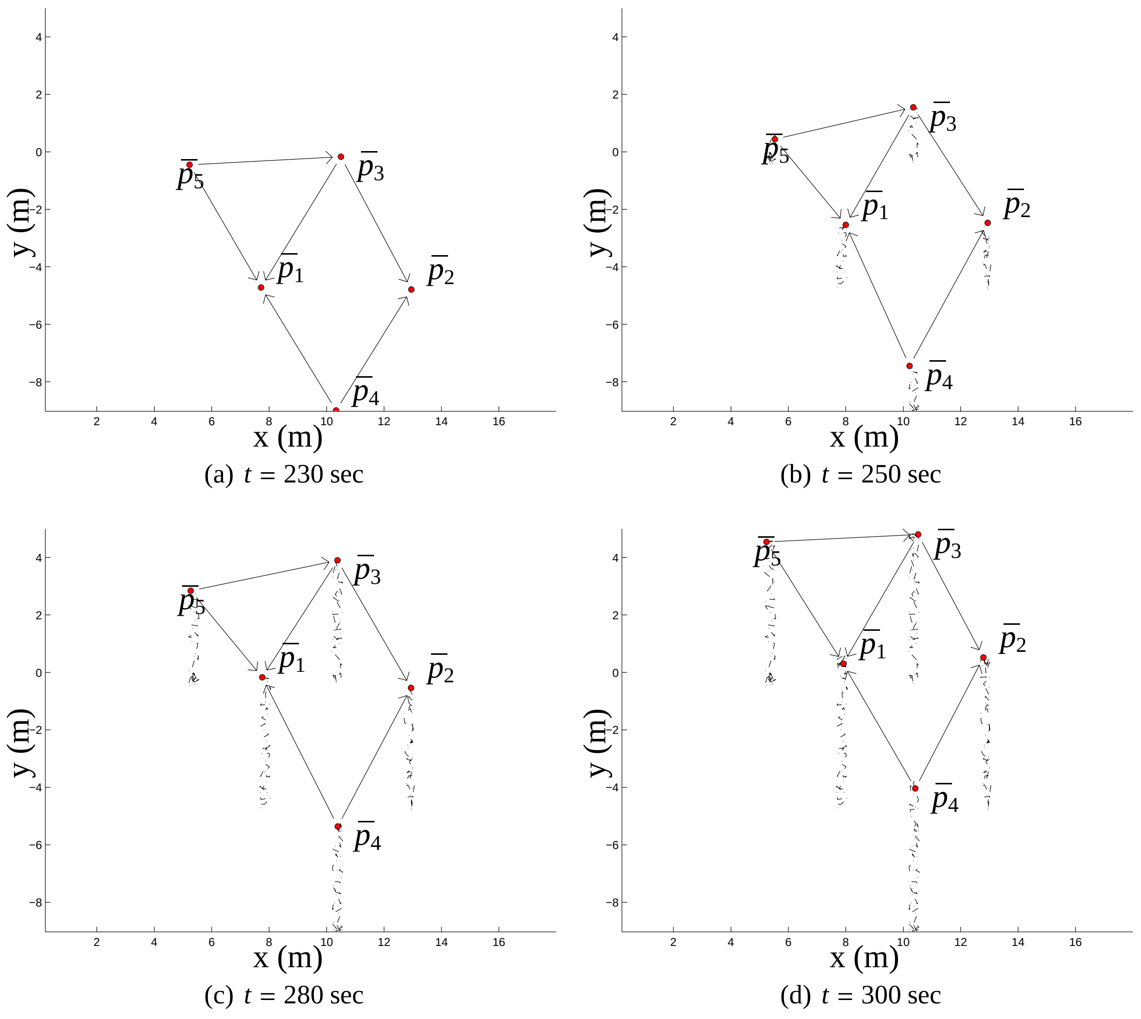

(a) t = 230 sec (b) t = 250 sec (c) t = 280 sec (d) t = 300 sec

Figure 7.12. Network trajectory during motion.

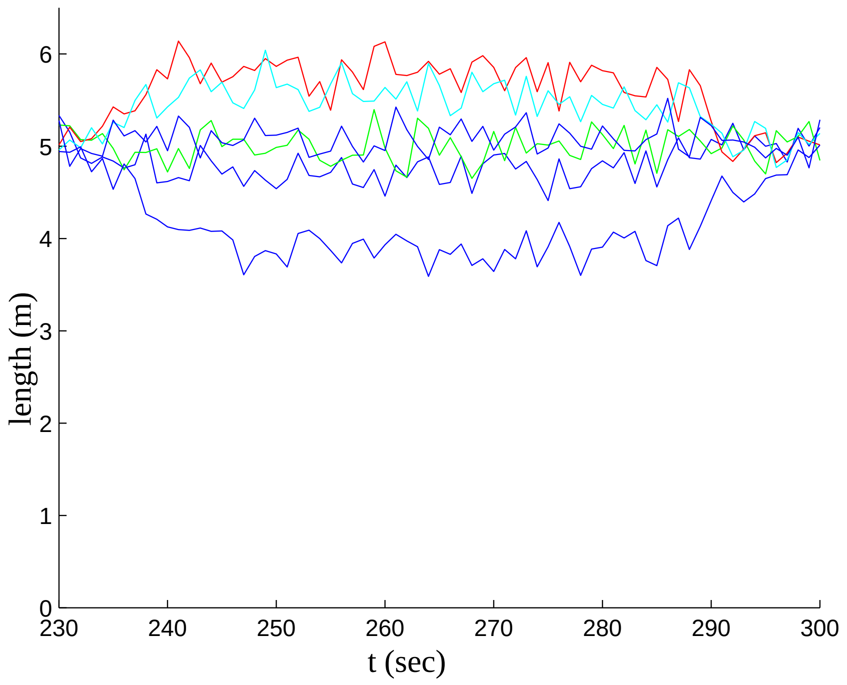


Figure 7.13. Network graph edges. Error is introduced during the formation motion. However, the edges stabilize back to 5 m when the formation stops moving.

CHAPTER 8
CONCLUSIONS

This work is part of project to deploy a multi-robot system as a wireless sensor network for meteorological data collection in Antarctica. Automatic tools for configuring and deploying the network are presented. These methods are tailored to the decentralized nature of the multi-robot network and the limited information available to each robot.

Multi-Robot Coordination with Embedded Graph Grammar Systems (EGGs)

In Chapter 3, we present preliminary work towards implementing triangulations of robots with a network of *SpiderMote* robots. These robots use an Embedded Graph Grammar (EGG) system, which defines the control laws for each robot, as well as how the robots sense, communicate, and switch modes. A communication protocol for multi-robot networks with global communication is introduced. This allows the for the implementation of the EGG system on the multi-robot network.

Rigid and Persistent Feasibility and Network Graph Generation

Chapter 4 presents methods for determining if a desired formation for the multi-robot network is *rigidly*, *persistently*, and *stably, persistently feasible* with respect to the limited information available to each robot. Given a *proximity range* which defines the maximum distance at which a pair of robots can sense/communicate with each other, these methods not only determine the corresponding feasibility, but generate an appropriate network graph for the formation. All rigid, persistent, and stably persistent network graphs are rigid. Persistent and stably persistent network graphs limit the number of constraints assigned to each robot to two or less. Stable persistent network graphs have the advantage of being acyclic. This facilitates the automatic definition of control laws for these graphs.

Control Laws for Multi-Robot Network Formations with Persistent Network Graphs

Using stable, persistent network graphs generated by the methods in Chapter 4, we present control laws for formations of mobile robots using these network graphs in Chapter 5. These control laws use circle-circle intersection solutions to determine the local geometry that they must satisfy with respect to their constraints in the network graph.

Formation Assembly with Embedded Graph Grammars (EGGs)

In Chapter 6, we present a method to automatically generate EGG systems for assembling formations of mobile robots. These generated EGG systems rely only on the relative locations of the robots. As such, the network does not need any localization to implement these EGGs. Further, this EGG system respects the proximity range limitation of the network. We demonstrate these EGG systems using a prototype network and present experimental results.

Deployment of Heterogenous Multi-Robot Networks with Embedded Graph Grammars (EGGs)

Chapter 7 presents a method for deploying heterogenous multi-robot networks, as well as an experimental demonstration of all of the methods in this work. The network is heterogenous in that only a proper subset of robots have localization ability. We present an EGG system that allows all the robots to navigate towards the initial deployment location. This is despite the fact that some network members cannot estimate their location relative to the initial deployment location. This chapter also includes experiments with a prototype multi-robot network. In these experiments, we use our methods for persistent feasibility, automatic stable, persistent network graph generation, automatic assembly EGG generation, the deployment EGG, and the control laws for multi-robot networks with persistent network graphs. This demonstrates how our methods allow a network with limited sensor and location ability to automatically achieve a feasible, user-defined network deployment automatically. This demonstration also includes implementation of all the methods presented in this work.

REFERENCES

[1] A. M. Howard, B. S. Smith, and M. Egerstedt, “Realization of the sensor web concept for earth science using mobile robotic platforms,” in *IEEE Aerospace Conference*, (Big Sky, MT), pp. 1–6, Mar. 2007.

[2] A. Howard, L. Parker, and B. S. Smith, “A learning approach to enable locomotion of multiple robotic agents operating in natural terrain environments,” *Intelligent Automation and Soft Computing*, vol. 14, no. 1, pp. 47–60, 2008.

[3] B. S. Smith, A. M. Howard, J.-M. McNew, and M. B. Egerstedt, “Multi-robot deployment and coordination with embedded graph grammars,” *Autonomous Robots*, vol. 26, pp. 79–98, Jan. 2009.

[4] B. S. Smith, M. Egerstedt, and A. Howard, “Automatic generation of persistent formations for multi-agent networks under range constraints,” in *Proceedings of the First International Conference on Robot Communication and Coordination*, 2007.

[5] B. S. Smith, M. Egerstedt, and A. Howard, “Automatic generation of persistent formations for multi-agent networks under range constraints,” *Mobile Networks and Applications*, 2009. To appear.

[6] B. S. Smith, J. Wang, and M. B. Egerstedt, “Persistent formation control of multi-robot networks,” in *Proceedings of the IEEE Conference on Decision and Control*, (Cancun, Mexico), pp. 471–476, Dec. 2008.

[7] B. S. Smith, M. Egerstedt, and A. Howard, “Automatic deployment and formation control of decentralized multi-agent networks,” in *Proceedings of the IEEE International Conference on Robotics and Automation*, (Pasadena, CA, USA), pp. 134–139, May 2008.

[8] B. S. Smith, J. Wang, M. Egerstedt, and A. Howard, “Automatic deployment of persistent multi-robot formations with sensing and localization limitations,” *IEEE Transactions on Robotics*, 2009. In submission.

[9] B. S. Smith, J. Wang, M. Egerstedt, and A. Howard, “Automatic formation deployment of decentralized heterogeneous multiple-robot networks with limited sensing capabilities,” in *Proceedings of the IEEE International Conference on Robotics and Automation*, (Kobe, Japan), May 2009. To appear.

[10] P. K. C. Wang, “Navigation strategies for multiple autonomous mobile robots moving in formation,” *Journal of Robotic Systems*, vol. 8, pp. 177–195, Apr. 1991.

[11] J. Desai, J. Ostrowski, and V. Kumar, "Control of formations for multiple robots," in *Proceedings of the IEEE International Conference on Robotics and Automation*, pp. 2864–2869, May 1998.

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

[13] J. A. Fax and R. M. Murray, "Graph laplacians and stabilization of vehicle formations," in *15th International Federation on Automatic Control (IFAC) Congress*, (Barcelona, Spain), 2002.

[14] H. G. Tanner, G. J. Pappas, and V. Kumar, "Input-to-state stability on formation graphs," *Proceedings of the 41st IEEE Conference on Decision and Control*, pp. 2439–2444, Dec. 2002.

[15] A. Das, J. Spletzer, V. Kumar, and C. Taylor, "Ad hoc networks for localization and control," in *Proceedings of the 41st IEEE Conference on Decision and Control*, vol. 3, pp. 2978 – 2983, Dec. 2002.

[16] J. Baillieul and A. Suri, "Information patterns and hedging brockett's theorem in controlling vehicle formations," *Proceedings of the 42nd IEEE International Conference on Decision and Control*, pp. 556–563, Dec. 2003.

[17] H. G. Tanner, G. J. Pappas, and V. Kumar, "Leader-to-formation stability," *IEEE Transactions on Robotics and Automation*, vol. 20, pp. 443–455, June 2004.

[18] J. A. Fax and R. M. Murray, "Information flow and cooperative control of vehicle formations," *IEEE Transactions on Automatic Control*, vol. 49, pp. 1465–1476, Sept. 2004.

[19] T. Eren, W. Whiteley, A. S. Morse, B. D. Anderson, and P. N. Belhumeur, "Information structures to secure control of globally rigid formations," *Proceedings of the 2004 American Control Conference*, vol. 6, pp. 4945–4950, 2004.

[20] T. Eren, W. Whiteley, B. D. Anderson, A. S. Morse, and P. N. Belhumeur, "Information structures to secure control of rigid formations with leader-follower architecture," *Proceedings of the 2005 American Control Conference*, vol. 4, pp. 2966–2971, June 2005.

[21] G. A. Kaminka and R. Glick, "Towards robust multi-robot formations," *Proceedings of the IEEE International Conference on Robotics and Automation*, pp. 582–8, May 2006.

[22] S. Sandeep, B. Fidan, and C. Yu, "Decentralized cohesive motion control of multi-agent formations," in *14th Mediterranean Conference on Control and Automation*, pp. 1–6, June 2006.

[23] 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), pp. 859–873, July 2006.

[24] L. Vig and J. A. Adams, "Multi-robot coalition formation," *IEEE Transactions on Robotics*, vol. 22, pp. 637–49, August 2006.

[25] N. Michael and V. Kumar, "Controlling shapes of ensembles of robots of finite size with nonholonomic constraints," in *Robotics Science and Systems*, (Zurich, Switzerland), June 2008.

[26] D. Y. Yeung and G. A. Bekey, "A decentralized approach to the motion planning problem for multiple mobile robots," *Proceedings of the 1987 IEEE International Conference on Robotics and Automation*, vol. 3, pp. 1779–1784, Mar. 1987.

[27] C. W. Reynolds, "Flocks, herds, and schools: A distributed behavioral model," *Computer Graphics*, vol. 21, pp. 25–34, July 1987.

[28] A. Jadbabaie, J. Lin, and A. S. Morse, "Coordination of groups of mobile autonomous agents using nearest neighbor rules," *IEEE Transactions on Automatic Control*, vol. 48, pp. 988–1001, June 2003.

[29] Z. Lin, M. Broucke, and B. Francis, "Local control strategies for groups of mobile autonomous agents," *IEEE Transactions on Automatic Control*, vol. 49, pp. 622–629, Apr. 2004.

[30] T. Balch and R. Arkin, "Behavior-based formation control for multirobot teams," *IEEE Transactions on Robotics and Automation*, vol. 14, pp. 926–939, Dec. 1998.

[31] W. Ren and R. Beard, "Consensus of information under dynamically changing interaction topologies," *Proceedings of the American Control Conference*, vol. 6, pp. 4939–4944, July 2004.

[32] G. Notarstefano, K. Savla, F. Bullo, and A. Jadbabaie, "Maintaining limited-range connectivity among second-order agents," *Proceedings of the American Control Conference*, pp. 2124–2129, June 2006.

[33] J. Cortes, S. Martinez, and F. Bullo, "Robust rendezvous for mobile autonomous agents via proximity graphs in arbitrary dimensions," *IEEE Transactions on Automatic Control*, vol. 51, no. 8, pp. 1289–1298, 2006.

[34] M. Ji and M. B. Egerstedt, "Distributed coordination control of multi-agent systems while preserving connectedness," *IEEE Transactions on Robotics*, vol. 23, pp. 693–703, Aug. 2007.

[35] M. M. Zavlanos and G. J. Pappas, "Potential fields for maintaining connectivity of mobile networks," *IEEE Transactions on Robotics*, vol. 23, pp. 812–816, Aug. 2007.

[36] M. M. Zavlanos and G. J. Pappas, “Dynamic assignment in distributed motion planning with local coordination,” *IEEE Transactions on Robotics*, vol. 24, pp. 232–242, Feb. 2008.

[37] H. Gluck, “Almost all simply connected closed surfaces are rigid,” in *Geometric topology, Lecture Notes in Math*, vol. 438, (Berlin), pp. 225–239, Springer, 1975.

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

[39] W. Whiteley and T. Tay, “Generating isostatic frameworks,” *Structural Topology*, vol. 11, pp. 21–69, 1985.

[40] 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.

[41] J.-M. McNew and E. Klavins, “Locally interacting hybrid systems with embedded graph grammars,” *Proceedings of the 45th IEEE Conference on Decision and Control*, pp. 6080–6087, 2006.

[42] J.-M. McNew, E. Klavins, and M. Egerstedt, “Solving coverage problems with embedded graph grammars,” *Hybrid Systems: Computation and Control*, vol. 4416, pp. 413–427, Apr. 2007.

[43] G. Laman, “On graphs and rigidity of plane skeletal structures,” *Journal of Engineering Mathematics*, vol. 4, pp. 331–340, Oct. 1970.

[44] D. J. Jacobs and B. Hendrickson, “An algorithm for two-dimensional rigidity percolation: The pebble game,” *Journal of Computational Physics*, vol. 137, pp. 346–365, June 1997.

[45] L. Henneberg, “Die graphische statik der starren systeme,” 1911.

[46] H. K. Khalil, *Nonlinear Systems*, ch. 4, pp. 179–180. Upper Saddle River, NJ 07458, USA: Prentice Hall, 3 ed., 2002.