Analysis and Implementation of Distributed Algorithms for Multi-Robot Systems

by

James Dwight McLurkin IV

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2008

Author .....
Department of Electrical Engineering and Computer Science
May 29, 2008

Certified by .....
Leslie P. Kaelbling
Professor
Thesis Supervisor

Accepted by .....
Terry P. Orlando
Chairman, Department Committee on Graduate Students

Analysis and Implementation of Distributed Algorithms for Multi-Robot Systems

by

James Dwight McLurkin IV

Submitted to the Department of Electrical Engineering and Computer Science
on May 29, 2008, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Computer Science

Abstract

Distributed algorithms for multi-robot systems rely on network communications to share information. However, the motion of the robots changes the network topology, which affects the information presented to the algorithm. For an algorithm to produce accurate output, robots need to communicate rapidly enough to keep the network topology correlated to their physical configuration. Infrequent communications will cause most multi-robot distributed algorithms to produce less accurate results, and cause some algorithms to stop working altogether. The central theme of this work is that algorithm accuracy, communications bandwidth, and physical robot speed are related.

This thesis has three main contributions: First, I develop a prototypical multi-robot application and computational model, propose a set of complexity metrics to evaluate distributed algorithm performance on multi-robot systems, and introduce the idea of the *robot speed ratio*, a dimensionless measure of robot speed relative to message speed in networks that rely on multi-hop communication. The robot speed ratio captures key relationships between communications bandwidth, mobility, and algorithm accuracy, and can be used at design time to trade off between them. I use this speed ratio to evaluate the performance of existing distributed algorithms for multi-hop communication and navigation. Second, I present a definition of *boundaries in multi-robot systems*, and develop new distributed algorithms to detect and characterize them. Finally, I define the problem of *dynamic task assignment*, and present four distributed algorithms that solve this problem, each representing a different trade-off between accuracy, running time, and communication resources.

All the algorithms presented in this work are provably correct under ideal conditions and produce verifiable real-world performance. They are self-stabilizing and robust to communications failures, population changes, and other errors. All the algorithms were tested on a swarm of 112 robots.

Thesis Supervisor: Leslie P. Kaelbling
Title: Professor

Acknowledgments

My financial support came from generous grants from Boeing. The SwarmBot robots were build at iRobot with grants from DARPA IPTO under contracts DASG60-02-C-0028 and N66001-99-C-8513.

There are many, many people to thank for help in getting to this point. First and foremost, Dara Bourne, my wonderful fiancée, has endured this entire process, and what the process does to the grad student. Thank you, Dara. (I owe you, big time.) My advisor, professor Leslie Kaelbling, has dispensed equal parts technical guidance, hard questions, patience, and even good advice. In many respects, I consider her to be the model advisor, and can only hope to be as good, and as useful, to my students. Finally, I am deeply grateful to my wonderful, encouraging, and loving family (and family-to-be) who has supported me throughout this process.

My committee, professors Erik Demaine, Daniela Rus, and Seth Teller, have provided all the things a committee should: insightful comments, obscure references, perplexed looks, and the occasional grumpy criticism. Their support has been invaluable. Ron Wiken is the man whos been showing me how to build stuff right since 1991. I wouldn't be the engineer I am today without his help and advice. The Swarm project team at iRobot included many special engineers, but the Swarm simply would not exist without Jim Frankel and Jennifer Smith. Thank you all.

Graduate students need each other. Aisha Walcott has provided huge support in reorganizing, rethinking, replanning, rewriting, and all the other "res" required when I'm going down the wrong path. Ed Olsen has been a constant source of technical advice, and the best "we've got to graduate and get a job" buddy a grad student could have. Mac Schwager has been my advisor on all things mathematical and control-theoretic, and the protagonist in many fascinating data-collection conversations of the form: "It's working!", "No, it's not.","It's totally working!", "That looks like a bug...", "Now, it's definitely working!","Hmmm, maybe...". I have been fortunate to be in a research group that is always willing to gather around the whiteboard to solve each other's problems. The "Brainy Ladies from G585" include: Sarah Finney, Meg Aycinena, Emma Brunskill, Natalia Hernandez Gardiol, Kaijen Hsiao, Luke Zettlemoyer and Nick Matsakis. (all brainy, some more lady than others)

Obviously, I'm too useless to do any *real* work, but fortunately, I've had the pleasure of working with many great undergraduate researchers over the years. All of the data collection infrastructure, huge amounts of analysis, finding contradictions in my theorems, and the soldering of little red lights to all 112 robots was done by hands other than my own. The credit goes to: David Blau, Brian Schmidt, Alexander Sanchez, Rian Hunter, Angelique Moscicki, Jeremy Smith, and Tony Valderrama.

Finally, I would like to thank the 112 little robots who have always done exactly what I have asked of them. Sometimes, every now and then, maybe just enough for a thesis, they do what I want.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Imagine searching for survivors in a collapsed building after an earthquake. Humans are either too big to search inside the debris, or too weak to lift the rubble to effect a rescue. But a multi-robot system comprised of thousands of ant-sized scouts, a dozen rat-sized structural engineers, and several brontosaurus-sized excavators would be ideal.

Imagine searching for fossils on Mars. A group of one thousand microrobots could survey a very large area, then request detailed analysis on promising discoveries from a more capable robot.

Imagine searching a building for an item of interest. A multi-robot system could start from a single location and disperse into the building, searching in all directions simultaneously to locate the object.

These scenarios are interesting because they all lend themselves to automation, but can be accomplished far better by multi-robot systems than by individual robots. These scenarios are hard because many open research problems lie between their solutions and the current state-of-the-art. In general, a multi-robot application will have four properties in common. First, there is a need for the agents to be highly mobile. If the problem can be solved with a system of stationary agents, then a multi-robot system is an overcomplicated solution. Secondly, the environments in which these systems operate could be larger than the range of a single robot's communication system, such as on the planet Mars, or might be communication-unfriendly, such as inside buildings. Therefore, multi-hop communications protocols will be required, and distributed algorithms that rely on communication to operate will have to respect bandwidth constraints while being robust to continually changing network topology. Third, the robots will need some geometric information about the positions of other robots, or they cannot coordinate their movements. However, we cannot always assume a centralized coordinate system, such as the global positioning system, will be available in the environment. Instead, we assume each robot has a sensor to measure the positions of other nearby robots, and we must design computational geometry algorithms that function when each robot has noisy measurements in its own coordinate frame. Finally, multi-robot distributed algorithms must be robust to population changes due to robot failures or the addition of new members.

These same application properties that make multi-robot systems interesting also present difficult research problems. Multi-hop communications networks, distributed algorithms, and robot configuration control are all active areas individually, and must be successfully combined in order to produce a multi-robot system. Geometric information about the positions of neighboring robots has to be used carefully, as measurements of the same

quantity from two different robots must be combined in a consistent way. Distributed algorithms must be self-stabilizing and be robust to population changes and robot failures. And multi-robot systems have different critical resources that limit performance, requiring performance metrics that include processing speed and memory usage, but also communications bandwidth and physical robot speed. Taken together, these problems present unique challenges to the multi-robot algorithm designer.

1.1 Contributions

This thesis has three main contributions: First, I develop a prototypical multi-robot application and computational model, propose a set of complexity metrics to evaluate distributed algorithm performance on multi-robot systems, and introduce the idea of the *robot speed ratio*, a dimensionless measure of robot speed relative to communication speed in networks that rely on multi-hop communication. I use this speed ratio to evaluate the performance of existing distributed algorithms for multi-hop communication and navigation. Second, I present a definition of *boundaries in multi-robot systems*, and develop new distributed algorithms to detect and characterize them. Finally, I define the problem of *dynamic task assignment*, and present four distributed algorithms that solve this problem, each representing a different trade-off between accuracy, running time, and communication resources.

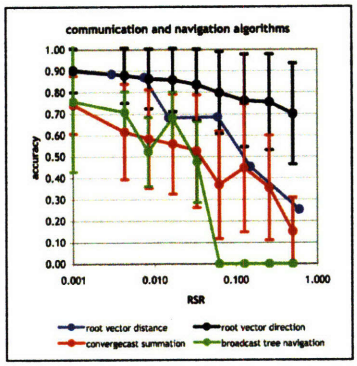
The core idea behind these contributions is the notion that algorithm accuracy, communications bandwidth, and robot speed are related. All of the algorithms I describe are self-stabilizing, meaning the robots will converge to the desired configuration from any initial configuration. Distributed algorithms rely on network communications to determine where to move the robots in the environment. This motion changes the network, which is, roughly speaking, a new input to the distributed algorithm. In order to achieve rapid convergence to an accurate result, robots will need to communicate quickly enough to keep their network up-to-date, so that the distributed algorithms will have current information to process. A network of fast-moving robots with slow communications will always be presenting stale inputs to their algorithms, which will degrade performance and produce less accurate results, and some algorithms might even stop working altogether. The *robot speed ratio* presented in chapter 4 captures one aspect of the trade-offs between communications, mobility, and algorithm accuracy.

The next few sections cover highlights of the contributions and provide a summary of the work presented in this document.

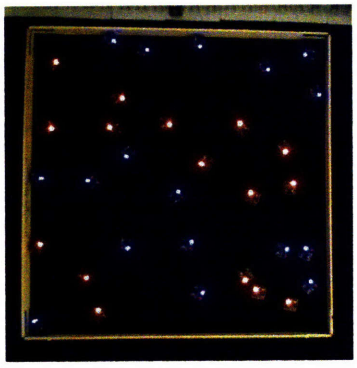
1.1.1 Applications, Models, Metrics and the Robot Speed Ratio

Multi-Robot Application Requirements: I describe features common to many multi-robot applications, then use these features to develop a practical model of multi-robot algorithm execution that incorporates the robots' computation, inter-robot communication, and physical configuration.

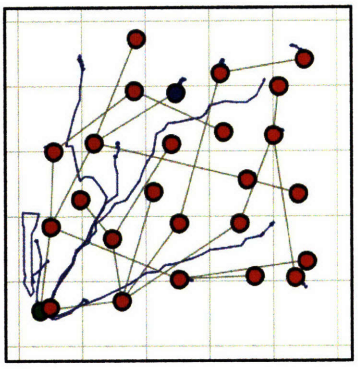
Complexity Metrics for Multi-Robot Systems: Algorithmic complexity measures characterize an algorithm's performance based on its use of physical resources, typically memory space and execution time. Complexity metrics for multi-robot systems must consider



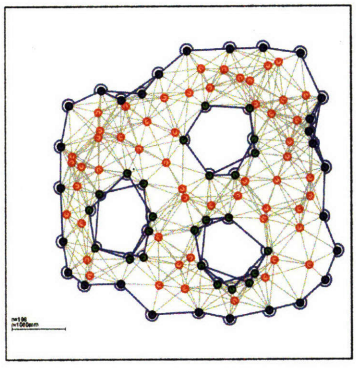
(a) Algorithm performance metrics



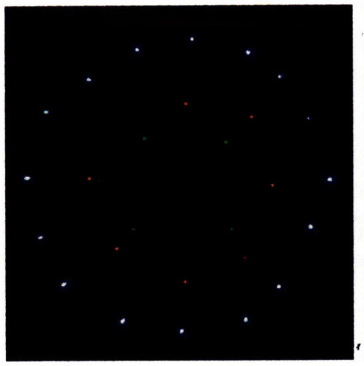
(b) Robots performing broadcast tree construction from root at lower left



(c) Experiment trace of broadcast tree navigation



(d) Boundary detection and sub-graph construction



(e) Robots detecting the external boundary (blue) and an internal void (green)

(f) Robots executing dynamic task assignment

Figure 1-1: The main contributions of this work. **a:** Algorithm accuracy and the *robot speed ratio* are platform-independent analysis techniques designed for multi-robot distributed algorithms. **b,c:** Ad-hoc communications networks and network-based navigation are fundamental algorithms in multi-robot systems. We quantify their performance using our metrics. **d-e:** We define, sketch a proof of, and verify the performance of distributed boundary detection and characterization. **f:** We describe four algorithms for dynamic task assignment, each with different performance trade-offs

a broader range of physical resources, such as robot mobility and network communications. These extra resources can often limit algorithm performance before processor speed or memory. To address this, I propose four additional complexity metrics for characterizing the performance of multi-robot distributed algorithms: physical accuracy, physical running time, communication complexity, and configuration complexity.

The Robot Speed Ratio: The robot speed ratio (RSR) is a dimensionless metric of robot speed in systems that rely on multi-hop communications. It is the ratio of robot speed to message propagation speed through the network. It is derived from known system parameters, and can be used at design time to trade off between them. For example, a designer can trade communications bandwidth for robot speed to maintain a desired RSR. We characterize algorithm accuracy in terms of the RSR, so if a designer requires a minimum accuracy, they can look up the maximum RSR. With this RSR, they can calculate the communications bandwidth required to operate at a desired robot speed, or vice-versa.

1.1.2 Boundary Detection

An algorithm that can determine the boundary of a multi-robot system has many potential applications. Robots on the boundary can estimate the perimeter of the configuration, devote more of their resources to surveillance, help prevent the network from disconnecting, or rectify voids in coverage. I develop a distributed algorithm to identify boundary robots, and two algorithms to estimate global boundary properties.

Distributed Boundary Detection: the Cyclic-Shape Algorithm I present a definition of a boundary suitable for use on a two-dimensional multi-robot system, and describe a distributed algorithm to compute it. The definition produces a boundary that captures convexities, concavities, and internal voids. It forms a contiguous subgraph of the robot network, so that messages can be routed around it to estimate global boundary properties.

Boundary Subgraph Construction and Global Boundary Classification: I present two algorithms to characterize global properties of the boundary. The first algorithm connects boundary robots into subgraphs, elects a single boundary leader on each subgraph, and propagates its ID to the other boundary robots. I provide a robust implementation of this algorithm, and a proof sketch that the external boundary subgraph forms a single connected component. The second algorithm determines whether a boundary is the external boundary or an internal void by aggregating local angular measurements on the boundary leader. The leader calculates the exterior angle of the polygon, determines the global boundary classification, then rebroadcasts this information to the other robots on the boundary.

1.1.3 Dynamic Task Assignment

Assigning robots amongst several different tasks is a requirement for many multi-robot applications. There are two parts to this problem: determining the correct distribution of robots to allocate to each task, and then assigning the robots to their specified tasks. We[1] address the second problem, and present four distributed algorithms to produce a desired

[1]Joint work with Daniel Yamins, originally published in June 2005.

task assignment. Each of these algorithms solves the same problem, but each represents a different trade-off in terms of communications usage, running time, and required network stability.

RANDOM-ASSIGNMENT The RANDOM-ASSIGNMENT algorithm assigns tasks to each robot randomly, weighted by the desired vector of task ratios. It runs quickly and uses no communication, but can have very large error from the desired assignment, especially in systems with small to medium numbers of robots.

SEQUENTIAL-ASSIGNMENT The SEQUENTIAL-ASSIGNMENT algorithm assigns tasks to individual robots sequentially, using minimal communications but a great deal of time.

SIMULTANEOUS-ASSIGNMENT The SIMULTANEOUS-ASSIGNMENT algorithm compiles a complete list of all the robots in the network on every robot simultaneously. It runs quickly, but uses a great deal of communication bandwidth to produce the list.

TREE-BASED-ASSIGNMENT The TREE-BASED-ASSIGNMENT algorithm is a compromise between SIMULTANEOUS-ASSIGNMENT and SEQUENTIAL-ASSIGNMENT balancing communications use and running time. However, it requires a stable network structure, which places limits on the mobility robots in order for the algorithm to run accurately.

1.2 Summary

For all the algorithms presented in this document, we provide proofs or strong arguments of correctness under ideal conditions, and present performance results from extensive testing on systems of 30 to 80 robots.

This document is organized as follows: Chapter 2 surveys related work. Chapter 3 describes multi-robot application requirements, our computational model, basic multi-hop communication algorithms, and our experimental setup. Chapter 4 describes complexity metrics for multi-robot algorithm evaluation, then uses these metrics and the multi-hop broadcast algorithm from chapter 3 to develop the robot speed ratio. Chapters 5, 6, and 7 describe communication and navigation algorithms, boundary detection algorithms, and dynamic task assignment algorithms, respectively. These chapters largely independent of each other, and the reader can skip to either after completing chapter 4. Chapter 8 are conclusions and directions for future work.

The next chapter presents an overview of related work, and situates my contributions with respect to the literature.

Chapter 2

Related Work

This thesis draws on a wide variety of literature from many different fields in computer science and robotics. Figure 2-1 illusturates the relationship between this thesis and the literature. We focus this discussion on work that is most closely related to the systems we study: large numbers of autonomous agents operating in two-dimensional environments with multi-hop communications networks. This chapter reviews the literature in general. Following chapters contain a detailed discussion of the work most pertinent to their content.

2.1 Distributed Algorithms

The core of any multi-robot system are the distributed algorithms that govern communication, configuration control, data processing, and state estimation. The three most important concepts this work uses are the idea of a synchronizer, self-stabilizing algorithms, and fault tolerance.

A *synchronizer* [4] is a construct that allows an asynchronous distributed system to be modeled as a synchronous distributed system. This greatly simplifies analysis, as a synchronous distributed system can be modeled as evolving through a series of discrete time steps. This does not necessarily require a shared global clock: I use local clocks and a system of rounds described in Chapter 3 to accomplish the same effect.

Because of errors in sensors and initial configurations, multi-robot systems require algorithms that are *self-stabilizing* [67]. A self-stabilizing distributed algorithm can take a system with arbitrary input and/or internal state, and through repeated execution, drive the system to a desired goal state. If there are more perturbations while the algorithm is running, it will correct these as well. Multi-robot systems use on this self-stabilizing property to handle changes in the environment and sensor errors. Our work applies self-stabilizing algorithms directly to multi-robot systems.

The two basic types of *fault tolerance* [67] are stopping failures, when a process (robot) stops functioning completely, and Byzantine failures, when a process produces arbitrary outputs. In this work, we must handle stopping failures in order to cope with population changes and robots with dead batteries. Since our algorithms are self-stabilizing, they are already equipped to handle this kind of failure. Byzantine failures and malicious processes, which can be modeled as a special case of Byzantine failures, are not a concern for this work. Because of the difficulty in dealing with these kinds of failures, we depend on cyclic redundancy checks to discard corrupt messages, and operating system checks to

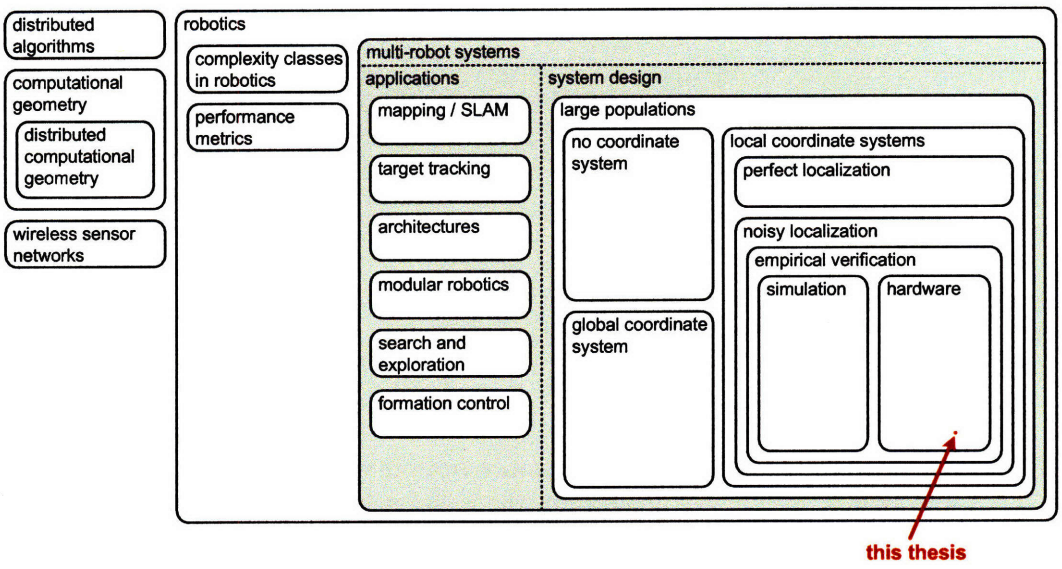


Figure 2-1: Diagram of relationships between the literature and this thesis.

shut down processors that are not behaving correctly. Ultimately, the best check for errant systems is a human user, and this is acceptable in a laboratory environment.

2.2 Computational Geometry

Computational geometry underpins many areas of robotics, and plays a key role in distributed algorithms for multi-robot systems. Using standard computational geometry algorithms on multi-robot systems presents two challenges. First, centralized algorithms must be redesigned as distributed algorithms. Second, some multi-robot systems only support geometric measurements relative to each robot, that is, each robot has its own *local coordinate system*. Since there is always uncertainty in sensor measurements, no two robots will have the same geometric data. Any algorithm that uses geometric information can reach different decisions on different robots. I believe that this "geometric consensus" problem is an important measure of the information requirements of multi-robot algorithms.

In this section, I describe four different classes of coordinate systems and their abilities and limitations. After that, I survey results in computational geometry.

2.2.1 Coordinate Systems

We divide models of coordinate systems into four groups based on two parameters. The first is whether measurements are made relative to a shared external set of axes, or on multiple axes relative to each robot. The second is whether or not the measurements are corrupted with sensor noise or not. In the descriptions below, I am referring to the coordinate system available to the robot the algorithm runs on, and I assume the geometric data being measured is robot positions.

1. Perfect Global Coordinates Traditional work in computational geometry assumes a perfect global coordinate system. Classic algorithms for convex hull, Delaunay triangulation, and line sweep algorithms [87,96] all require that the positions of all the robots are known to a single processor and these positions are measured on the same global axes. Most algorithms assume that an edge can be placed between any two robots, regardless of the distance between them. However, interest in ad-hoc networks has sparked much work on unit-disk graphs [60,61] (but Kuhn also proposes an extended model [59] which includes variable radii at an additional cost in routing operations), which are graphs with a maximum edge length of one. It is common to assume these graphs are connected, but not fully connected, see Poduri [94] for analysis on connecting disconnected groups of robots. For algorithms that require all positions to perform computation (centralized computation), all the robots can learn the positions of all other robots using $O(n^2)$ messages.

While this model seems unrealistic, it is appropriate in describing the positions of modules in a physical implementation modular robot. [102,118] If two modules are connected, it is reasonable to assume the the mechanical interlocks secure the modules into a known, fixed, rigid relationship. The topology of the modules can be measured exactly in a body-coordinate system. Note that this is not the same as measuring the pose of the resulting configuration if the modules are articulated; it is just measuring the relationship between connected modules.

Even though perfect measurements are not achievable on any physically realizable system, often times the uncertainty is small and centered around the actual position, as is the case with a global positioning sensor. Because of this, many applications that actually have noisy global coordinates can be reasonably well modeled by theory that assumes perfect global coordinates.

2. Noisy Global Coordinates This a common and practical coordinate system model on multi-robot systems. The uncertain global pose estimates could be derived from many different sources. A GPS system provides noisy estimates centered around the actual position of the robot. A probabilistic SLAM-style localization algorithm [35,110] could provide a locus of possible pose estimates. Measurements from coordinate systems can be combined to form an estimate of global position [17,44,76,79], but this can be difficult, and sometimes require high node degree, or be susceptible to errors from voids in the connectivity.

As with perfect global coordinates, if each robot needs to know the positions of all the others, all the robots can learn the positions of all the other robots using $O(n^2)$ messages. Even though each robot has some error in its measurement, by sharing this information with their neighbors, there will still be consensus amongst neighbors, and a globally consistent geometric graph from the unions of the local neighborhoods.

3. Perfect Local Coordinates Problems in the sensor networks have prompted a great deal of new work in *distributed computational geometry* [7,98] that addresses systems that operate on local neighborhoods, and with local coordinates; *i.e.*, with positions to neighboring robots that are measured in each robot's coordinate frame.

There are many variants on the unit-disk construction with different types of geometrical information, those with full pose $\{x, y, \theta\}$ estimation to nearby robots, those with distance information only, those with connectivity information only, Yao graphs, Gabriel graphs [66,114], and NET graphs [93].

With perfect local coordinates, geometric quantities measured on robot a will be consistent with those measured on any other robot. For example, robot a's measurement of the distance to robot b, robot b's distance measurement to robot a, and robot c's measurement of the distance between a and b will all be identical. An algorithm that uses geometric information to make decisions will make the same decisions on all three robots, and the unions of the local neighborhoods produces a globally consistent geometric graph.

4. Noisy Local Coordinates This is the coordinate system model used in this work. Each robot measures geometric information (*e.g.* the positions of its neighbors) with some uncertainty, and no two robots will have identical measurements of the same geometric data. For example, robot a's measurement of the distance to robot b will not be the same as robot b's distance measurement to robot a, or robot c's measurement of the distance between a and b. This uncertainty means there is no consensus of measurements between neighbors, and any algorithm that uses geometric information to make decisions can make different decisions on all three robots.

This makes some tasks, like computing a globally consistent Delaunay triangulation, difficult. It seems the only way to address this problem is to have the robots reach a consensus on their geometric measurements. This distributed geometric consensus problem could force some robots to accept network geometry inconsistent with their local measurements. Even worse, these kinds of "sensor overrides" might have to propagate throughout the entire group of robots.

Another approach would be to generate a noisy global coordinate from the noisy local coordinates using one of the approaches mentioned above. However, it is not clear how these approaches would fare on a highly dynamic system with bandwidth limits on communication. To my knowledge, such a system has not been empirically tested.

2.2.2 Relative Power of Coordinate System Classes

This last class of coordinate systems is separated from the first three by a distributed consensus problem. This lack of consensus would seem to make this coordinate system less powerful than the others. Using the dominance relation defined by O'Kane and LaValle [85] we can can order the relative power of the different kinds of coordinate systems a multi-robot system can use. They define a *dominance relation*:

$$\text{system}_2 \trianglerighteq \text{system}_1,$$

means that system_1 can simulate the behavior of system_2, and is therefore a more "powerful" system. The $\equiv$ operator indicates that two systems are equivalent in power, and the $\cup$ operator allows two different system's properties to be merged. We use this operator to indicate what would be required to make two systems equivalent. Applying this to the four coordinate systems from above, we have:

$$\begin{aligned}
\text{Perfect Global Coordinates} &\trianglerighteq \text{Noisy Global Coordinates} \trianglerighteq \text{Noisy Local Coordinates} \\
\text{Perfect Global Coordinates} &\trianglerighteq \text{Perfect Local Coordinates} \trianglerighteq \text{Noisy Local Coordinates} \\
\text{Perfect Global Coordinates} &\equiv \text{Perfect Local Coordinates} \cup O(n^2) \text{ communications messages} \\
\text{Noisy Global Coordinates} &\equiv \text{Noisy Local Coordinates} \cup \text{Distributed Consensus}
\end{aligned}$$

As the $\trianglerighteq$ is transitive, noisy local coordinates are the least powerful of the four systems, and limits the computational power of the robots the most. I believe that this distributed consensus problem (or, making a noisy global coordinate system from noisy local coordinates) is the most important distinction between algorithms in computational geometry when applied to multi-robot systems. While there are approaches to solving this problem, it is unclear if they are sufficiently frugal with communications and computation or nimble enough to maintain accurate results in a highly dynamic configuration. The work I present in this thesis does not try to solve this problem, but instead explores what can be accomplished without geometric consensus.

2.2.3 Algorithms

Unit-disk graphs are a well-studied construction in computational geometry. The unit-disk model is ill-suited for the radio communications systems of sensor networks [117] because radio energy is strongly affected by the surrounding environment, especially when the antenna are close to the ground. The radios on our experimental robots can have link quality transition from 99.99% to 20% with a few centimeters of displacement of the robot, or even a nearby grad student.

However, unit-disk models are a good model for the geometric localization system that a multi-robot system is likely to have (we make this assumption formally in the next chapter). Kuhn and Li construct spanners unit-disk graphs, and compare their performance to Yao graphs, and discuss generating an approximate Delaunay triangulation from perfect local coordinates [60,64,70] Wang and Li describe a distributed algorithm for constructing a bounded planner spanner of a unit-disk graph [65,113]

Poduri and Ghrist use a *neighbor every theta* (NET) graph, a topology in which the angle between the neighbors of any node is less than a parameter theta. This construction can be used to design topology control algorithms [93] and determine whether a node is inside or outside a cycle in the graph [39].

The Delaunay triangulation and Voronoi diagram have many uses. Cortes [23], Schwager [104], and many others compute a *centroidal Voronoi configuration* of robots. These algorithms run by moving robots to the (often weighted) center of mass of their Voronoi cell. This requires individual robots to be able to determine their own Voronoi cells, and some algorithms require the robots to have a global compass to determine the orientation of their cell. The triangulation is also used to efficiently compute the Alpha-shape of a set of points [29]. The alpha shape algorithm is the inspiration for the boundary detection work in Chapter 6.

Guibas et. al. describe *Kinetic data structures* [6,75], geometric data structures with bounded update complexity when used on graphs with highly dynamic topology. They allow invariants to be conserved with minimal messaging in the face of dynamic network

changes.

Some of these techniques can be directly applied to multi-robot systems with noisy local coordinates, but most cannot. Each must be considered carefully to see if there is a requirement for consensus between neighboring robots. In this work, I rely on brute-force reconstruction of broadcast trees by continuously rebroadcasting messages from the root, so obviate the need to minimize the messages required to cope with mobile nodes. Many of the unit-disk models assume limited geometric information to be more compatible with radio systems, but on a robot platform, we can assume a more complete set of geometric information, in particular between neighboring robots. Robots with local coordinate systems are able to compute a local approximation to their Voronoi cell, which can be used in centroidal Voronoi algorithms.

2.3 Wireless Sensor Networks

Wireless sensor networks have many similarities to multi-robot systems, in particular with respect to communications and networking, and the style of distributed algorithms that are practical.

The analysis of the limits of performance and congestion in multi-robot systems can be very similar to that in ad-hoc networks [77, 115]. Broch takes into account the speed of the mobile nodes [14], an idea that I extend to define the *robot speed ratio* in Chapter 4. Since a multi-robot system will have geometrical information about neighbor placement, geographic routing protocols can be useful [91].

However, a fundemental difference between routing in ad-hoc networks [22, 80] and sensor networks and multi-robot systems is the assumption that a named destination is known to the sender of a message. This is not always true, and distributed algorithms often broadcast information and reply to whomever needs it, instead of routing specific messages to specific destinations. The oft-cited papers by Intanagonwiwat and Estrin [31, 50] introduced *directed diffusion* which is a system of broadcasts and responses suitable for these "not-quite-named-destination" routing requirements. The broadcast trees I use in this work are the simplest form of ad-hoc multi-hop communication, and use only a fraction of their framework.

Ultimately, sensor networks exist to collect data for users. Madden et. al. introduced the notion of database-style queries on sensor networks with TinyDB [68, 69], which runs on the very popular operating sensor node operating system, TinyOS.

Similar to sensor networks except in population size, an amorphous computing system [1] is a distributed system in which very large numbers of simple computational elements are spread throughout space. Imagine mixing Avogadro's number of computational elements into a can of paint, then painting a wall with it. Particularly inspiring pieces of work are Coore's Ph.D. thesis [18] on drawing arbitrary 2-d shapes with a distributed programming language, and Nagpal's Ph.D. thesis on programming a sheet of "smart paper" to fold itself into origami [78].

While the sensor network community is very similar in terms of hardware to multi-robot systems, there are some important system distinctions that affect algorithmic choices. The first is that robots move autonomously and sensor nodes don't. This affects the types of algorithms that the two communities study, search/exploration vs. aggregate/process/route. Also, the energy requirements of mobility on macro-scale robots can far exceed the power used in communication and processing, making these resources free. In this work, we as-

sume the robots are communicating constantly. In contrast, energy in a deployed sensor network is often the overriding system constraint, and nodes are powered off most of the time to conserve battery life. These differences separate the two fields, but there is still sufficient similarity to share ideas and even code.

2.4 Robotic Complexity Analysis

Understanding how much information a robot needs to perform a task was studied by Erdmann [30], and how different tasks and robots can be compared in a rigorous way was studied by Donald, then O'Kane and LaValle [27,83–85]. These ideas are an important tool in robotics research. We used the notion of a dominance relation earlier in this chapter to indicate which coordinate systems are more powerful than others. However, these rigorous techniques do not allow a user to calculate expected robot performance from system parameters.

2.5 Multi-Robot Systems

For useful (albeit dated) surveys of multi-robot systems see articles by Parker [89] and Cao et al. [16]. I borrow some of their application taxonomy, but focus this discussion on the areas that are most closely related to the systems we study: large numbers of autonomous agents operating in two-dimensional environments with dynamic, multi-hop communication networks. I also add a second hierarchy of multi-robot system design, and discus the closest related work to this document.

2.5.1 Applications

There are many applications for multi-robot systems. In this work, we focus on applications that involve large numbers of robots operating in a two-dimensional environment that is larger than their communication range. This is a common scenario for mapping, search and exploration,

Search and Exploration Multi-robot systems are particularly well-suited for search and exploration, and there is a tremendous amount of work on the subject. Many approaches use heuristics or behavior-based [15] approaches to disperse robots throughout their environment [3,8,9,47] Hybrid systems of robots and sensor networks use robots to distribute and repair arrangements of sensor networks [19,20,46]. Other approaches use a centroidal Voronoi algorithm to disperse robots [23,104]. To survey this literature carefully would require an entire paper, and it does not directly support what we present here. In this work, it is important to keep in mind that search and exploration is one of the most useful task for multi-robot system, and bias our algorithms and metrics toward such applications. Given a a choice between two different, but similar techniques, we choose the one that is better suited for search and exploration.

Formation control Formation control is second only to search and exploration in the amount of work in the multi-robot community. Although not a robotics paper, the "Boids" [99] program of Reynolds is an early example of an effective distributed formation

control algorithm. Behavior-based approaches [5, 36] use heuristics and potential fields to hold robots into formations guided by a designated leader. It is not clear if this type of approach can scale to large populations of robots. A more control-theoretic point of view [25, 26] produces similar results, but with similar concerns for large populations.

A less constrained type of formation is a crystalline structure. Spears and Spears at. al. develop a notion of *physicomimetics* that uses robots to mimic physical models of particles [42, 106–108]. This allows them to develop an analytical model of the potential energy wells required to form a stable crystalline formation, and use this to program simulated and real robots.

The least constrained type of formation is to coalesce robots into larger groups to form a single connected component, with no concern for relative placement. Poduri et. el. describe simple heuristics to achieve coalesce in a bounded environment [94,95], and develop an analytical model to predict upper and lower bounds on convergence time.

Decision Making Making global decisions from local data is a key component in multi-robot systems. The simplest form of a global decision is a leader election algorithm [67], in which robots use unique IDs and the network to determine a global leader. Bertsekas and Tsitsiklis [111] and others [86] describe *agreement algorithms* where the multi-robot system can reach a global consensus on a particular quantity. The results are powerful, and prove consensus even in networks with limited connectivity, like unit-disk graphs, and in changing topology, like multi-robot systems. Bishop and Klavins [12] describe a different approach in which robots take local measurements of a global quantity, than all converge to the same conclusion. Robots can also measure local quantities, like physical interference [40], to determine when there too many robots in a given area or performing a given task. in this way, local decisions can produce more efficient global behavior.

Modular Robotics Modular robotics is an important area within multi-robot systems. They come in many different configurations, from the generic module systems of Yim and Rus [56, 102, 103, 118, 119], to more purpose-built designs with impressive wall-climbing abilities [92], to disconnected pieces of computation that depend on random motion from their environment to find each other and connect [54].

However, the models of networking and mobility can be very different from the systems we focus on. Modules that are physically connected can share messages via wires, eliminating the need to share communications bandwidth and the strict constraints that imposes on disconnected multi-robot systems. Also, physically connected modules have strict constraints on the configurations they can assume, as they must remain connected as they move. These two distinctions make them very different from planar multi-robot systems.

Manipulation Multi-robot manipulation work falls into two catagories. Capture-based approaches [101, 105, 109] where multiple robots surround a larger object and move it by contacting the object and moving themselves. These robots do not have specialized effectors for manipulation, they "form" a manipulator by their physical configuration around the object.

This is in contrast with more standard robotic manipulation techniques [88, 116] where robots move small objects around the environment with traditional grippers. These tasks include foraging for materials, and construction of simple shapes in the environment.

Navigation Using sensor networks as navigational cues is a new form of robot navigation that is particularly well-suited for multi-robot systems. Different approaches vary somewhat [11,24,63,73], but they all share a common theme. There is one distinguished goal node that spreads a message throughout the network. Mobile navigating robots use this message to "route" themselves to the goal node, using the network neighbors as guides. I evaluate a version of this algorithm in Section 5-8.

Coordinate Systems Constructing noisy global coordinates from noisy local coordinates has been the subject of much work. Approaches range from using broadcast trees to measure distances through the network [79]. Roumeliotis and Bekey describe a distributed Kalman-filter based approach [100] that allows the equations to be computed locally on each robot, with small update messages between nearby robots. The approach allows for anchoring to control integration of error, but it is not clear how it would scale to a large system of robots. Grabowski and Khosla use trilateration and carefully controlled motion to localize a group of five small robots [43]. More recently, Moore et. al. [76] developed a system using "stable quads" that produces good localization from range measurements only, but requires a high average vertex degree.

2.5.2 Multi-Robot System Hardware

Working with physical hardware has the benefit of providing a high-fidelity model of the errors present in a multi-robot system. Inter-robot communications, neighbor localization, and sensor errors can be difficult to model accurately in simulation packages. In this section we look at recent multi-robot hardware platforms, and the key design parameters they employ.

Dudek et. al. proposed a taxonomy for swarm robots [28] that is still a good formalism for classifying platforms. We provide a very brief summary of their classifications here, please refer to the reference for complete descriptions:

- **Swarm Size:** The size of the population: alone, pair, multiple, infinite.
- **Comm. range** The communication range: none, nearby, infinite.
- **Comm. topology:** The topology of the communication network: broadcast, address, tree, graph.
- **Comm. bandwidth:** The cost of communication (bandwidth): $\ll$ movement, $\approx$ movement, $\gg$ movement, no comms.
- **Reconfigurability:** How can the swarm physically reconfigure: static, coordinated, dynamic (robots can go anywhere).
- **Coordinate System:** The coordinate system available to the robots: (from above) noisy global, noisy local, none.
- **Composition:** The types of robots: homogeneous or heterogeneous.
- **Unit Processing Ability:** The kind of computation available at each robot: Turing machine on all platforms.

Name, Location	Swarm Size	Comm. range	Comm. topology	Comm. bandwidth	Reconfigurability	Coordinate System	Composition
Ants, Nature	infinite	nearby	broadcast	≈ movement	dynamic	noisy local	species-dependant
Nerd Herd, MIT, Mataric92	multiple	infinite[1]	broadcast[1]	≈ movement[1]	dynamic	noisy global	homogeneous
Alice, EPFL	multiple	infinite[2]	broadcast[2]	» movement	dynamic	none[2]	homogeneous
Centibots, multiple[4]	≈ infinite	infinite	broadcast	« movement	dynamic	noisy global	heterogeneous
Swarm, iRobot, McLurkin04	≈ infinite	nearby	broadcast	≈ movement	dynamic	noisy local	homogeneous
e-puck, EPFL	infinite	nearby[3]	broadcast[3]	≈ movement	dynamic	none	homogeneous
AmigaBot, multiple[5]	≈ infinite	infinite	broadcast	« movement	dynamic	noisy global	heterogeneous
USC	multiple	infinite	broadcast	« movement	dynamic	noisy global	heterogeneous
Clodbuster III, U. Penn	multiple	infinite	broadcast	« movement	dynamic	noisy local	homogeneous

[1] Communications was through a centralized hub, and could be made to simulate nearby or infinite

[2] There are upgrades to this robot that support communication and localization, but none currently supports both

[3] With Zigbee radio turret upgrade

[4] SRI, Stanford, U. Washington, ActivMedia

[5] SAIC, University of Tennessee, Telcordia Technologies, University of Southern California

Table 2.1: Extant multi-robot taxa.

Table 2.1 shows the classifications of existing multi-robot platforms. I add an additional axis, the type of the coordinate system, which is one of the four types from the discussion above: perfect global, noisy global, perfect local, noisy local, or none. All the systems reviewed in this work have Turing machine equivalent processing.

We can group the robots largely into two classes: those that can make a geometric map of their environment, and those that cannot. Robots that are able to map their environment typically use a laser scanner sensor to build a map and determine their pose. These robots are currently fairly large, and can easily carry a high-bandwidth radio with a range that covers a typical laboratory environment. The coordinate system is a "noisy global" system, as each robot can measure its position relative to its map, then share maps with the other robots in the environment. [34,43,100] However, the amount of computation and communication involved in this kind of approach makes it seem unlikely that it will scale to large (100 or more) numbers of agents.

The non-mapping robots are typically smaller, mostly custom-built, and have harsh limitations on sensing and processing. The biggest limitation of these platforms is a sensor to localize the robot in the environment, or to localize a robot's neighbors. Of those listed, there are three robots capable of localizing their neighbors. The Clodbusters from University of Pennsylvania use an omnidirectional camera to track nearby robots. The AmigaBots from SAIC and University of Tennessee [48,49] use an interesting system of visual fiducials mounted on a stalk above each robot. This design lets individual robots determine the full pose, $\{x, y, \theta\}$ of neighboring robots. The SwarmBots used in this work use a custom-built infrared communications system that determines the pose of nearby robots.

2.6 Summary

Now that we have described the multi-robot literature, the next chapter describes multi-robot application requirements that affect system design, and what assumptions I believe will not hold in a practical application. We use these application requirements to develop a practical computational model, and review common communications algorithms.

Chapter 3

Assumptions, Computation Model, and Communications

What is the canonical multi-robot application? It would be a task that requires mobile agents, or a sensor network would be a more appropriate solution. It would be a task that cannot be performed by a single robot. It would be a task where the environment makes it impossible to use a centralized command and control architecture, forcing the robots to rely on local communication and control algorithms. In this chapter, we develop a prototypical application to define the scope of the problems we consider, allowing us to specify system constraints, design goals, and an appropriate model of computation.

The first section enumerates the application requirements carefully, and describes their consequences for mobility, communications and sensing. To these requirements and consequences we add several key assumptions to fully define our problem space and construct an appropriately abstracted model of computation. The second section describes this computational model of a multi-robot system and its associated communications network. The third section reviews existing algorithms for multi-hop communication that we use in this work. The last section describes our experimental setup for algorithm evaluation and data collection.

3.1 The Canonical Multi Robot Application

In this section, we propose four fundamental requirements that we believe any multi-robot application will have, and motivate how these requirements constrain the system design and define the scope of problems we consider. The four requirements of our canonical multi-robot application are: highly mobile agents, local inter-robot communication, robot position estimation, and robustness to population changes. Finally, we use these requirements to produce a list of assumptions that will let us define our model.

3.1.1 Multi-Robot Application Requirements

Requirement 1: Multi-Robot Applications Require Mobility. We focus our discussion on applications that require many, if not all, of the robots to be mobile most of the time. Applications such as search and rescue, exploration, and mapping are ideal for swarms of mobile robots, and are the types of problems to keep in mind as we design our algorithms. Applications that do not require mobility are better suited for distributed sensor networks,

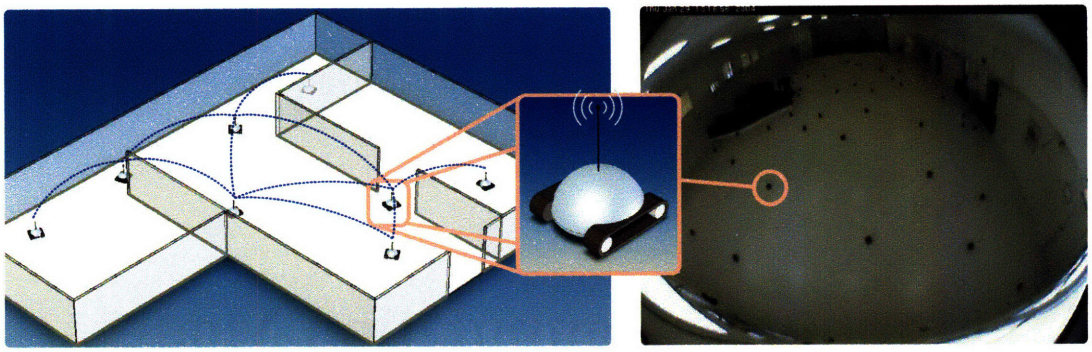


Figure 3-1: An example multi-robot application is building exploration. The illustration on the left shows a concept of exploration, and the picture on the right shows a fish-eve view of a team of 104 robots (not all are visible in this image) exploring a building, using methods described in this thesis.

which have a different set of assumptions and design constraints. Applications that have a finite amount of mobility, such as a sensor network that self-deploys, then remains stationary, can be modeled as a multi-robot system during the mobile phase, then as a sensor network during the sessile phase.

In systems where it is impractical for the robots to carry enough energy reserves for their entire mission, we assume that each robot in the network must remain within range of an energy supply of some sort. This could be a charging station, or a larger robot with greater energy reserves. The location of the energy source will limit the maximum range of the individual robots, but still allow us to assume that the system has access to sufficient energy to support unconstrained mobility up to this maximum range.

Requirement 2: Multi-Robot Applications Must Use Local Communications. We assume that many applications will require the robots to cover an area that is larger than an individual robot's communication range, *e.g.*, exploring the planet Mars, or that the robots are deployed into an environment that severely limits communication range, *e.g.*, inside a building. In order for the robots to share information beyond their immediate neighbors, multi-hop communications protocols will be required to transport data around the network.

But even if it is possible to design a communication system with sufficient range to allow all the robots to communicate directly with all the other robots, such a network would cause the communication bandwidth requirements on each robot to grow as $\Omega(n)$ bits/second, where n is the total number of robots in the system. To see this, we must also assume that inter-robot communications uses a half-duplex, shared channel, which is true for any practical system of wireless communication. Therefore, any time one robot sends a message to any other robot, *all* other robots receive it, and cannot transmit while they are doing so. If every robot has some information to communicate, then all robots must have sufficient bandwidth to receive messages from all the other robots, producing the $\Omega(n)$ bandwidth lower bound. This type of network will only be practical to implement in very small populations. In this work, we assume that individual robots cannot accommodate local communications bandwidth that is at any time $\Omega(n)$. Note that this means robots can

neither send $\Omega(n)$ messages per unit time, or any message of size $\Omega(n)$ at any time.

We assume instead that each robot can handle communications bandwidth that is around $O(m)$ bits/second, where m is the number of neighbors. Assuming regular communications with m neighbors (we will justify this assumption shortly), if each robot sends a message of size $O(1)$ bits, each robot will need sufficient bandwidth to receive $O(m)$ bits/second. Sending messages to neighbors of neighbors would require messages of size $O(m)$ bits, and bandwidth of $O(m^2)$ bits/second. In general, it is practical to assume bandwidth of $O(m^k)$ bits/second, where k is a small integer. On our experimental platform, $k = 2$ is a realistic limit.

Requirement 3: Multi-Robot Applications Require a Sensor to Measure Other Robot Positions. A *configuration control* algorithm is one that modifies the physical arrangement of the robots relative to each other. There are many algorithms of this type, including navigation, formation control, and density management. In order to run any configuration control algorithm, each robot requires geometric information about the positions of other robots in the system. A global positioning system can provide this information, but this kind of system will not be available in many of the applications where swarms would be useful - inside buildings, underground, and on other planets. Additionally, we do not assume access to an infrastructure that can provide the robots with a shared global coordinate system, or that robots can localize themselves relative to places they have previously visited, or even that the robots' odometry is reliable enough to make a short-term map.

Instead, we assume that each robot has some way to determine the pose, $p = \{x, y, \theta\}$, of other nearby robots relative to its own local coordinate frame. All robot-level and group-level positioning is based on this *local neighbor pose estimation*. We do not specify how this information is gathered, but it is important to assume that pose measurements taken on different robots will have independent errors. This means that geometry that should be consistent across multiple robots, *e.g.* three robots in a triangle measuring each other's poses, will actually be inconsistent due to this sensor noise. This lack of global consensus on local pose estimates is a critical feature of multi-robot systems that must be carefully handled when designing algorithms that use and share geometrical information.

While off-the-shelf radio systems provide very low-quality geometric information about their communications links, there are many practical ways each robot could measure the positions of its neighbors. A sensor to measure the positions of other robots could be vision-based [48], using passive or active fiducials to locate nearby robots. The cricket system uses a combination of radio and sonar [97] to accurately estimate the range of communication links. Some of the earliest autonomous robots used structured optical emitters and detectors to locate each other. [112] Our experimental platform uses the spiritual successor of the third approach, with a custom system of infrared emitters and detectors [73] on each robot that can measure neighbor poses in the receiving robot's local coordinate frame. Robots with access to a global positioning system would be able to simulate local coordinates and run a superset of the algorithms we consider.

Requirement 4: Multi-Robot Applications Require Algorithms that are Robust to Population Changes. An advantage to using a multi-robot system is that individual robots can be expendable, so long as the algorithms are *scalable* and *self-stabilizing*.

A scalable algorithm is one that is insensitive to the population of the system. However, many useful algorithms, such as leader election, require that information propagate

throughout the entire network. If we assume a uniform density of robots arranged in a somewhat circular [1] configuration, the diameter of the physical arrangement will grow approximately as $\sqrt{n}$, and for a fixed communication radius r, so will the diameter of the communications network. For our purposes, we will call an algorithm scalable if its performance grows as a sublinear function of n. This allows us to consider algorithms where the performance scales as a function of the diameter of the configuration of robots.

A self-stabilizing algorithm will reconverge to the desired configuration after an arbitrary number of robots are removed from or added to the population, so long as the network remains connected after the removal. We use this to boot-strap a new set of robots into a stable initial configuration, and to respond to changes in population. In addition to large, infrequent population changes, our charging station assumption from above also implies that there will be a constant low-level flux of robots leaving and re-entering the network. The internal state of any robot entering the network will be out of sync with the rest of the group, and all algorithms must be designed to accept any number of robots with arbitrary state and evolve their state to assume the appropriate task and location within the group. Note that this assumption handles stopping failures, *i.e.*, the total failure of any number of robots, but does not handle Byzantine failures, *i.e.*, robots that exhibit random or malicious behavior [67].

Consequence 1. Mobility and Local Communications: Frequent Communications. If communications are local and robots are mobile, then moving robots must announce themselves when they move to a different location to discover new neighbors, and the discovered neighbors must reply so that they can be detected by the mobile robots. If many of the robots are moving, which is true for the class of problems we consider, then it can be more efficient to have all the robots regularly broadcast an announce message and eliminate the responses. With a system of regular announcements, handshaking between two robots can be achieved by including robot b's handshake response to robot a in robot b's next announce message. For example, robot a broadcasts its announcement to all of its neighbors, and expects an acknowledgement from any, or all, of its neighbors. Any neighbor that needs to reply to a can append the reply to its next announcement, assuming the two robots are still within communications range. Grouping all the messages from a single robot into a single announcement reduces the chance of replies causing collisions on the shared communications channel, but adds more latency to handshaking.

Depending on the range of the communications system and the speed of the robots, network links may remain viable only for a limited number of messages. The exact number of messages that can be sent between two robots depends on the size of the messages, the inter-robot bandwidth, the two robot's velocity vectors, and the communication range. These parameters determine whether the ad-hoc network formed by the robots' communications is fairly stable over time and can be constructed carefully, or is highly dynamic and must respond quickly to network topology changes.

The analysis of multi-robot distributed algorithms can be simplified greatly if all the robots transmit their announcements with a shared fixed period, τ, but with different individual offsets to prevent message interference. This defines a local *round* of computation; a period of time in which each robot receives an announcement message from each of its neighbors, processes these messages, and then transmits its own announcement message

[1] We can define "somewhat circular" as any configuration where the ratio of perimeter to area is approximately $\frac{2}{r}$, the same as a circle.

in a burst at the end. Globally, if we examine the state, C, of all the robots at a time t, wait one round duration, τ, to time $t' = t + \tau$, then re-examine the new configuration, C', we can be certain that each robot has completed one round of computation. This creates a global synchronizer, and allows us to model the asynchronous multi-robot execution as a single synchronous global execution over a series of discrete rounds. This allows a much simpler computational model to accurately capture the behavior of the entire system. We will assume periodic transmission for the rest of this work.

Sensor network applications need to limit radio activity to prolong battery life. Mobile robots in a multi-robot system must communicate frequently. A system of robots that are constantly moving and communicating will use energy rapidly. We do not address power utilization or conservation in this work, but again assume that the robots have sufficient energy reserves or access to an external energy source to allow unconstrained communication.

Consequence 2. Local Neighbor Pose Estimation and Local Communications: Local Network Geometry. We assumed that each robot can determine the relative pose of nearby robots, and that each robot can communicate with robots within a fixed communication range. In our algorithms, robot b is a neighbor of robot a if a can communicate with b and a can determine b's relative pose. We call the combination of neighbor communication and relative pose estimation *local network geometry*. In this work, we assume the network connectivity in local network geometry relationships are reciprocal; if robot a can communicate with robot b, then robot b can communicate with robot a. However, since the relative pose estimates have local errors, the geometric relationships are not symmetric; a's estimate of b's pose is different from robot b's estimate of robot a's pose. The local network geometry forms the foundation for all of the algorithms presented in this work.

Consequence 3. Local Network Geometry and Maximum Range from Energy Source: Network Rooted to Energy Source. Our first assumption was that robots are highly mobile and must carry sufficient energy reserves for their entire mission, or be constrained to remain within range of an external energy source. In systems where the latter is true, the robots must have some way to navigate to the energy source. However, the local network geometry does not tell a robot about the location of anything beyond its immediate communications range. Without a map of the environment, the robots must rely on multi-hop network connectivity to form a network routing path which the robots can use for physical routing. This constrains the entire network to be *rooted* to the energy supply. The network must remain connected to the energy source to provide a navigable path for individual robots to follow. We describe a multi-hop navigation algorithm suitable for this in Section 5.6.

To summarize our requirements from above: We consider the class of multi-robot applications that require that **most of the robots are mobile most of the time** and that there is **sufficient energy for unconstrained mobility and communication**. We assume **local communications between neighboring robots** using a **half-duplex shared communications channel** with sufficient bandwidth to support a **maximum message size of** $O(m)$, where m is the maximum number of neighbors of any robot.

Each robot needs geometrical information about the positions of its neighbors in order to move relative to them, and we assume that **each robot can determine the pose,**

$p = \{x, y, \theta\}$, **of its neighbors relative to its own local coordinate frame**. We further assume that **pose estimates have local error**, so that no cycle of robots will produce a set of measurements between them that is globally consistent.

We assume that our **algorithms are self-stabilizing and robust to population changes**. This is required for boot-strapping a network into an initial configuration, to handle large, infrequent population changes, and to deal with low-level population flux from robots leaving to and returning from charging stations.

We realize that moving robots with limited communication range must communicate frequently to find new neighbors. We make the stronger assumption that **each robot broadcasts announcements to its neighbors at periodic intervals with a shared fixed period**. We can use this fixed period to define a global round of computation; a period of time after which all robots have communicated, updated their internal state, and broadcast their announcements to their neighbors. We combine the communications network and local pose estimation from neighboring robots to produce the **local network geometry**, which is the foundation of all the algorithms we present.

These application assumptions and consequences arise from the class of problems we consider, and from practical hardware constraints. We use these assumptions to define the class of multi-robot algorithms we consider in this work. These constraints also differentiate multi-robot distributed algorithms from distributed algorithms in general, ad-hoc networking, and distributed sensor networks. This makes distributed multi-robot systems a unique algorithmic environment, in particular with respect to the algorithm execution cycle where computation guides mobility, mobility affects network topology, and network topology affects the next round of computation.

3.1.2 Additional Assumptions

We make several more assumptions to fully define our problem. These assumptions help us develop practical algorithms and a model with manageable complexity for analysis of multi-robot systems.

We assume that the network formed from the local communications between robots forms a **single connected component**. Since we assume that all communications are local, a disconnected network would have no way to relay messages from one component to the other, and therefore could not execute a distributed algorithm. We also limit our work to cases where the entire group of **robots are on the same 2-dimensional surface**.

We require that each robot have a **unique id**. At the low level, this allows the networking and localization protocols to discriminate between messages from nearby robots. At the algorithmic level, unique IDs are required to break symmetry in bounded time. [67] From an engineering perspective, unique IDs are trivial to produce, and can either be programmed into the robots during their initial software download, or built into the hardware with a unique ID chip.

We assume that the robots are running **homogeneous software**. This is not much of a constraint, as any single program can have many different modes. We also assume that all the robots have **homogeneous hardware**. This ensures that all robots are fungible; any robot can do the task of any other. This would likely be the first assumption to be challenged in a practical application, as heterogeneity among sensors, actuators, and other physical and computational abilities might be desired, or even required, to achieve success in a particular application.

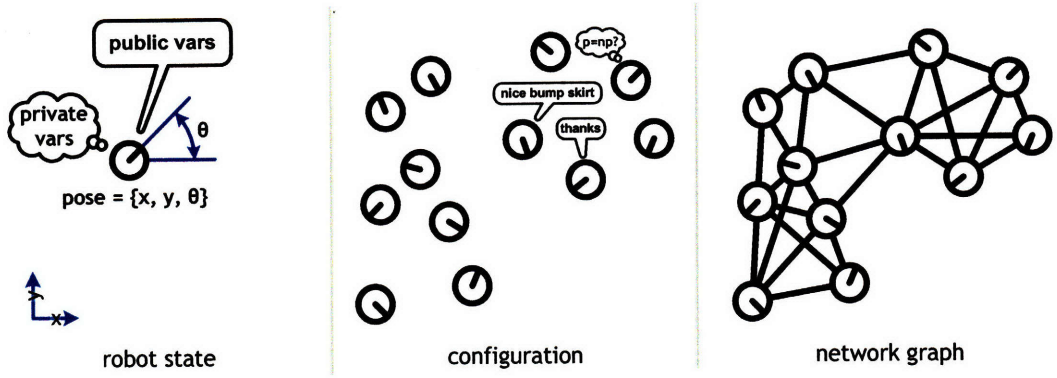


Figure 3-2: The *robot state* is the tuple of its global pose measured in an external reference frame and private and public variables, $state_a = \langle \text{ID}_a, \text{pose}_a, \text{privateVars}_a, \text{publicVars}_a \rangle$. A *configuration* is a group of robots. The configuration is valid if the *network graph* induced by connecting all robots whose separation distance, $d < r$, is connected.

These assumptions define the problem we consider and allow us to construct a model of computation that matches the practical constraints of multi-robot systems. The next section describes our model.

3.2 Multi-Robot System Model

In this section we define a model for multi-robot algorithm analysis. There are four parts to this model, the *robot state* of each robot, the *configuration* of a group of robots, the *network graph* that connects the configuration, and the *algorithms* that transform one configuration to another.

3.2.1 Robot State

We define the state of an individual robot, a, as the tuple of its unique ID, its global pose measured in a shared external reference frame, and its private and public variables,

$$state_a = \langle \text{ID}_a, \text{pose}_a, \text{privateVars}_a, \text{publicVars}_a \rangle$$

Note that the robot does not have access to its pose, but it can modify it by moving, or store it by remaining stationary. There are n total robots in the network, where $n \leq n_{max}$ for some finite integer n_{max}. Each robot has a unique identification number, a processor, and enough memory and computational resources to store and operate on $O(n_{max})$-sized data structures.

3.2.2 Multi-Robot Configuration and Configuration Network Graph

We define a *configuration*, C, as the collection of the states for all n robots in the network. Each robot can communicate with neighbors within a fixed radius r of its location. Without loss of generality, we can let $r = 1$. This produces a geometric unit-disk graph, G, in which each robot is a vertex and the communications links between robots are edges. We refer

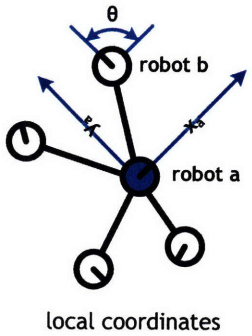


Figure 3-3: Robot a can estimate the pose $p_{ab} = \{x_{ab}, y_{ab}, \theta_{ab}\}$, of robot b relative to its own coordinate frame. We assume that there is some error in each of these estimates. This means that robot b's estimate of robot a's pose, $p_{ba} = \{x_{ba}, y_{ba}, \theta_{ba}\}$, will not be consistent with p_{ab}, as they will have different errors.

to this graph G as the *configuration network graph*. This graph is used for all inter-robot communication. We require the graph G to always be connected, as a graph composed of multiple disconnected components cannot share information globally and therefore cannot function as a group. We say the configuration C is *valid* if its graph G is connected.

We do not assume that the robots can localize themselves in a global external coordinate frame. This makes it difficult to guarantee that the robots can keep themselves grouped into a single connected component, or regroup if separated by some distance $d > r$. A disconnected swarm will run the same algorithm on each separate component. Practical configuration control algorithms make every effort to keep the swarm from disconnecting. If the swarm should disconnect, and happen to reconnect, any algorithm must tolerate these changes in population and produce a consistent single configuration from the previously disjoint configurations. Therefore, a disconnected swarm that fortuitously reconnects will eventually continue executing the algorithm correctly.

Each robot a can estimate the pose to a neighbor, say b, $p_{ab} = \{x_{ab}, y_{ab}, \theta_{ab}\}$ relative to its own coordinate frame, as shown in 3-3. Each component of the pose measurement has some sensor error with a distribution centered around the actual value. We will assume that this error in the pose estimate is small enough to allow reasonably accurate relative robot positioning and algorithm execution. However, the existence of these pose estimation errors has important algorithmic consequences, namely, the pose of robot b measured in robot a's coordinate system will never be symmetrical the pose of a measured in robot b's coordinate system.

3.2.3 Multi-Robot Algorithm

We model an algorithm A as a transformation from any valid configuration C_1 and its associated graph G_1, into a new valid configuration C_2 with graph G_2, such that C_2 satisfies a given set of properties. These properties can be physical, for example, having disperse to a uniform density, or computational, for example having computed the global sum of a quantity measured on each robot. The algorithms in this work (save one) are all deterministic, but still have a range of final configurations that satisfy the desired properties. For example, a uniform density dispersion property does not specify anything about the

particular physical arrangement of the robots, so any arrangement that achieves the desired density is acceptable. In addition, sensor, communication, and navigation errors add noise to the execution which must be tolerated by allowing a range of valid configurations during operation and at termination.

Therefore, in our computational model, an algorithm A does not produce a unique C_2 and G_2, but instead maps the input configuration to a range of valid output configurations, all of which satisfy the specified property. We can define the correctness of an algorithm by evaluating whether all possible resulting configurations satisfy the desired set of properties. However, the range of acceptable output configurations makes a binary measure of correctness a limited metric. There are typically some configurations that are "more correct" than others, requiring a scalar metric to quantify the physical accuracy of algorithms. We discuss these new metrics in the next chapter.

3.3 Network Communications

Network communications are a fundamental component to distributed algorithms on multi-robot systems. There are three basic types of network communication algorithms used in this work: one-hop communications, broadcast tree communications, and convergecast communications. Each of these communication techniques is progressively more complicated and builds upon the previous one. Algorithms use a combination of these techniques to create a *global communication structure*, which is discussed in more detail in Section 4.3.

3.3.1 One-Hop Communication

One-hop communication is used for local inter-robot messaging and is the foundation of all other communication. Each robot broadcasts its externally visible (public) variables at the end of every round, *i.e.* after every τ seconds. Each robot's first announcement occurs at a different random offset, $\Delta < \tau$, after a global start time, t_0. All subsequent messages are sent τ seconds after the previous one. The random initial offset produces a simple local time-division multiple-access (TDMA) [51] channel-sharing scheme similar to the Aloha protocol [2]. However, since we assume the robots have communications bandwidth that is less than $O(n)$, it is not possible to assign each robot a globally unique TDMA transmission slot. From the point of view of a single robot, announcements from its neighbors will be uniformly distributed over its local round, reducing the chance of message collisions, but not totally preventing them. If neighboring robots communicate at the same time, their messages will collide in the physical medium and both will be corrupted and discarded by the receiving hardware.

To preserve the fixed round duration critical for algorithm analysis, we d not employ a random back-off scheme, but instead rely on the robots' mobility to change the network topology, preventing two robots that transmit at the same offset from having their messages collide indefinitely. Additionally, we rely on the user, or a configuration control algorithm, to limit the robot density to avoid congestive network failure [77]. In practice, this system works well unless the robot density is high. With our experimental setup, networks with average vertex degree of less than around 10 provide good performance. The implementation of the low-level communication system has a maximum capacity of 16 neighbors per robot, which provides more than enough neighbors for our algorithms. However, the neighbor localization system uses the same communications hardware to

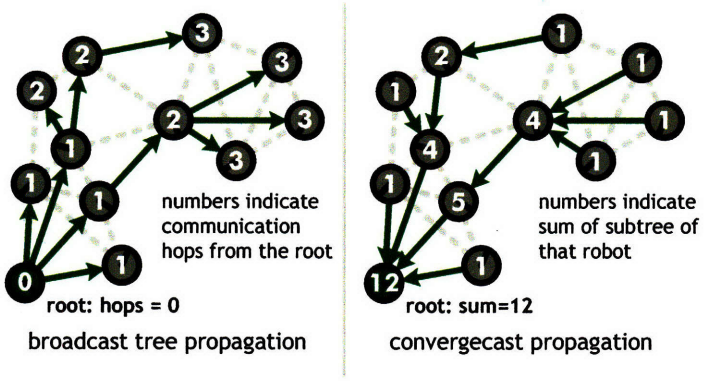


Figure 3-4: Broadcast tree communications and convergecast communications. Broadcast tree communications builds a minimal spanning tree as the message propagates from the root. This creates a directed, acyclic message propagation graph. This is the predominant mode of communication for the algorithms in this work. Convergecast communications use a broadcast tree as a routing structure to relay messages back towards the root. While useful to compute global quantities, it requires a stable tree to route over, which limits its utility in rapidly changing networks.

determine the pose of the nearby robots, where collisions and message loss cause errors in the position estimates of the neighbors. This affects the performance of algorithms that rely on local network geometry, such as the boundary detection algorithms described in Chapter 6.

Because the physical area around each robot is a shared communications resource, it is desirable for each robot to transmit all the announcement messages from one round of computation in a burst to minimize interference with other nearby communications. To abstract this low-level periodic communications infrastructure from the high-level distributed algorithms, we use a publish/subscribe system of shared public variables between neighboring robots. An algorithm on robot, a, can write to its public variables asynchronously, with no concern for when the current round ends. The communication system transmits the current state of all of robot a's public variables in a burst at the end of the current round. Each neighbor, say robot b, receives the messages, and updates its local copy of robot a's state. Any algorithm on robot b can query this saved state of the public variables from robot a at any time. This simplifies the implementation of distributed algorithms, as they can run asynchronously with respect to the communication system, but always have access to the most recent data. However, in the proofs of the algorithms, it is occasionally convenient to imagine that discrete messages are sent at specific places in the code. Usually, this departure from the actual implementation does not affect the analysis, but this must be treated carefully.

3.3.2 Broadcast Tree Communications

A broadcast tree is a multi-hop messaging procedure used in many routing protocols to find routes through *ad hoc* networks [50, 90]. An illustration of broadcast tree communication is shown in Figure 3-4. This algorithm is described carefully in [73], we provide

a summary here. A *root* robot creates a message, $msg = \langle hops, rootID, timestamp, data \rangle$, and makes it public to all of its neighbors. The *hops* field contains the number of communication hops the message has traveled from the root, the *rootID* is the unique ID of the root, the *timestamp* is a non-decreasing counter on the root that aids in message cleanup, and the *data* is the actual data payload of the message. In general, each robot in the configuration will have a copy of the message from the root, although local copies will have their hops adjusted to reflect the distance traveled in the network. There can be many different types of messages from many different roots propagating simultaneously in an algorithm, $msg_1 \ldots msg_k$, where k is the number of message types. Each message type is handled independently.

At the end of every round, for each message type, each robot collects k lists of m messages, one list per type, and one message from each neighbor. The robot increments the *hops* field in each received message to indicate the additional communication "hop" the message has just traveled. Once the lists are complete, each one is inspected by a *selection criterion* to select a single message from this list to store and republish. If no messages meet its selection criterion, then the robot will republish its previous message, but only for p additional rounds. This message "persistence", p, helps cope with communication errors. In our implementation, $p = 4$ works well.

The selection criterion for each message type determines how the messages will propagate by removing messages from the list in a series of steps. After all the steps are complete, any remaining messages meet the criterion, and any one of these can be selected to republish[2]. The neighbor the selected message came from becomes the parent on the broadcast tree.

For each type, given a list of messages and the previously stored message , the selection criterion for a normal broadcast tree is as follows:

1. Retain messages on the list that have traveled the fewest number of *hops*, discard all others.

2. Of the remaining messages, retain messages on the list with *hops* $\leq$ (*hops* of stored message), discard all others.

3. Of the remaining messages, select any message to store.

For example, if robot a is currently publishing a message with 3 hops, and receives a list of messages with 1, 2, and 3 hops, it will select one of the 1-hop messages to republish as a 2-hop message. If it receives messages with 2 and 3 hops, it will select one of the 2-hop messages. If all of the messages it receives have 3 hops, it will not select any message from the list, and retransmit it current message until it times out. At that time, it can select a message with a larger number of hops.

When robot a selects a message to republish, the sender of that message becomes robot a's parent on the broadcast tree. The selection criterion above always selects the message that has traveled the fewest number of hops from the root. This forces the propagation to proceed away from the root of the message, producing a directed, acyclic spanning tree, T, of the graph G. The total computation of the tree construction requires $O(diam(G))$ rounds.

[2]In practice, selecting messages randomly can lead to undesirable thrash in the network structure, as robots can select a new message from a new parent on each round. I use several more layers of heuristics to break ties to try and keep the network as stable as possible.

Broadcast tree communication can be used to implement a *leader election* algorithm [67] by using an alternate selection criterion. For each message type, given a list of messages and the previously stored message , the selection criterion for a leader election broadcast tree is as follows:

1. Retain messages on the list with the lowest value of *rootID*, discard all others.
2. Of the remaining messages, retain messages on the list that have traveled the fewest number of *hops*, discard all others.
3. Of the remaining messages, retain those with *hops* $\leq$ (*hops* of stored message), discard all others.
4. Of the remaining messages, select any message to store.

This leader election criterion is similar to the previous one, except that messages from a root with a lower ID will be selected over those from a root with a higher ID, even if the message from the higher ID root has traveled fewer hops. This criterion uses the broadcast tree to perform a leader election, so that after $diam(G)$ rounds, the only messages remaining in the network are those from the root with the lowest ID, which is the leader. As above, this criterion also produces a directed, acyclic spanning tree of he graph G, rooted at the robot with the lowest ID.

The tree is continually rebuilt from the root outward; if the root is removed, the gradient dies out in a controlled fashion using the *timeStamp* field. To generate a controlled decay, the root robot increments the *timeeStamp* field of its message each round. This continuously increasing number can be used by individual robots to determine if a message being purveyed to them by a neighbor is newer than the one they are currently storing. The selection criterion for a normal broadcast with a controlled decay is an extension of the first criterion for a normal broadcast tree:

1. Retain messages on the list with *timeStamp* $>$ *timeStamp* of stored message, discard all others.
2. Of the remaining messages, retain messages on the list that have traveled the fewest number of *hops*, discard all others.
3. Of the remaining messages, retain messages on the list with *hops* $\leq$ (*hops* of stored message), discard all others.
4. Of the remaining messages, select any message to store.

The first step removes messages with older *timeStamps* than the stored message; selection then proceeds as with a normal broadcast tree. If the root robot is removed from the network, all the robot will eventually have messages with the same *timeStamp*, and will not accept old messages from their neighbors to store. After p rounds, the tree will start to decay, but in a controlled fashion, starting at the one-hop robots, then the two-hop robots, and so on, until the stored messages on all the robots have timed out. Without the *timeStamp*, messages in a rootless tree propagate forever, with an ever-increasing number of hops. This deconstruction also takes $O(diam(G))$ rounds.

The self-stabilizing properties of these algorithms are essential for applications in swarms of mobile robots. They allow the communication structure to adapt to changing network topologies and population changes. Broadcast trees are the primary technique for disseminating information throughout the multi-robot system.

3.3.3 Convergecast Communications

Convergecast is used to propagate information towards the root of a broadcast tree T, using the broadcast tree structure as a routing table. [68, 69] However, individual messages are not routed directly towards the root of the tree, as this would require inter-robot communications bandwidth of capacity $O(n)$, especially for robots near the root. Instead, intermediate processing combines messages from subtrees, then sends this partial result towards the root via parents in the tree. This ensures the communications bandwidth required of each robot is still $O(1)$, but requires the tree to remain fairly stable for $depth(T)$ rounds to allow the messages to converge back onto the root.

An example of a convergecast is an algorithm to count the total number of robots in the network, illustrated in the right-hand panel of Figure 3-4. It is best to imagine the process starting from the leaves of the tree. Leaves send messages containing the count of their subtrees to their parents. The count from a leaf is 1, as leaves only have one robot, themselves, on their subtree. Their parents compute the sum from all of their children, add one for themselves, then pass this information up the tree to their parents. This process repeats each round on all robots at all levels of the tree, and after $depth(T)$ rounds, the root receives an accurate sum of the current population as long as T remains stable during this process. The total running time is $t = 2 \cdot depth(T)$ rounds, because the initial broadcast tree construction requires $depth(T)$ rounds to complete, and the convergecast requires an additional $depth(T)$ rounds.

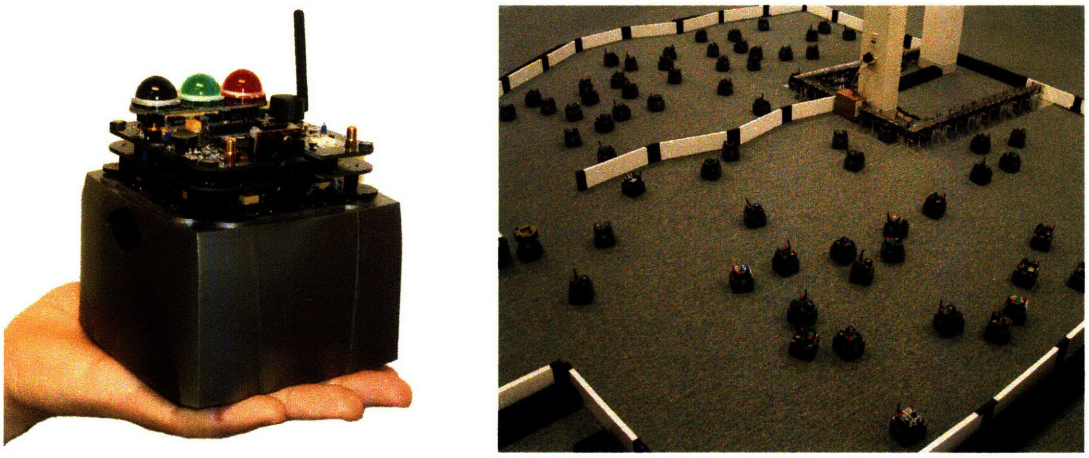
We can use a similar convergecast algorithm to estimate the depth of the tree. This can be used to place an upper bound on the diameter, as $diam(G) \leq 2 \cdot depth(T)$. Leaves label themselves as *level 1* robots. Every other robot computes its level by taking the maximum of the levels of each of its children, and adding 1. This repeats each round on all robots at all levels of the tree, and after $depth(T)$ rounds, the root robot will have computed its level correctly. This level is the maximum depth of the tree T. This algorithm is more robust to network changes than summation, as there are potentially many paths that the maximum level can use to get back to the root.

3.4 Experimental Apparatus

All of the algorithms described in this thesis were validated on a swarm of 100 robots. The experimental setup included a data logging system to record telemetry from the robots during executions, and a vision-based positioning system that provided ground truth for analysis.

3.4.1 Robots

The SwarmBot [52] robot platform was used to validate the performance of the distributed algorithms. The robots are a 12 cm cube. Each robot is fully autonomous, using only local computation and sensor readings to run the algorithms. Each robot has a processor with an ARM 7TDMI core running at 40mhz with 3MB Flash and 1MB RAM. Each robot has a unique ID chip. There is a bump skirt for low-level obstacle detection and wheel encoders to measure odometry. The robots have four wheels and use skid steering to turn. This, plus their small wheelbase, means that the integrating odometry measurements to estimate pose produces very poor localization results.



(a) A SwarmBot

(b) The Swarm.

Figure 3-5: **a.** Each SwarmBot has an infra-red communication and localization system which enables neighboring robots to communicate and determine their pose, $\{x, y, \theta\}$ relative to each other. The three lights on top are the main user interface, and let a human determine the state of the robot from a distance. The radio is used for data collection and software downloads. **b.** There are 112 total robots in the Swarm, but a typical experiment uses only 30-50 at a time. There are chargers shown in the upper right of the picture.

The main sensor used in this work is the infra-red communication and localization system. This system lets nearby robots communicate with each other and determine their pose, $p = \{x, y, \theta\}$ relative to each other. The system has a maximum localization and communication range of around 2.5 meters, but all experiments were run at the lowest transmit power setting, which has a range of about 1.0 meter. The lowest power setting is used to produce multi-hop networks within the confines of our experimental workspace, which is an 2.43 m $\times$ 2.43 m (8′ $\times$ 8′) square. The system can determine range and bearing to neighboring robots with an accuracy of 2 degrees and 20 mm when there is 500 mm of separation between robots. Because the system needs two messages from a neighboring robot to make a pose estimate, the accuracy of the localization degrades when there is a high density of robots and message collisions increase. Since the system is line-of-sight, nearby neighboring robots can occlude communications from more distant neighbors. The data rate of the system is 250 Kbit/s, but packetization and protocol overhead reduce effective throughput to around 100 Kbit/s, which must be shared with all robots within range.

The SwarmBot has custom hardware dedicated to the centralized user interface. This includes the three large LEDs on the top of the robot and the audio system. These devices let a human determine the state of the robots without having to look away from them to a computer screen. There are 108 total different blinking patterns, but only a small fraction is used in any one experiment. [74] Each robot has a 1 Mbit/s radio, which is used to download new programs remotely and for data collection. The data collection system is discussed in the next section.

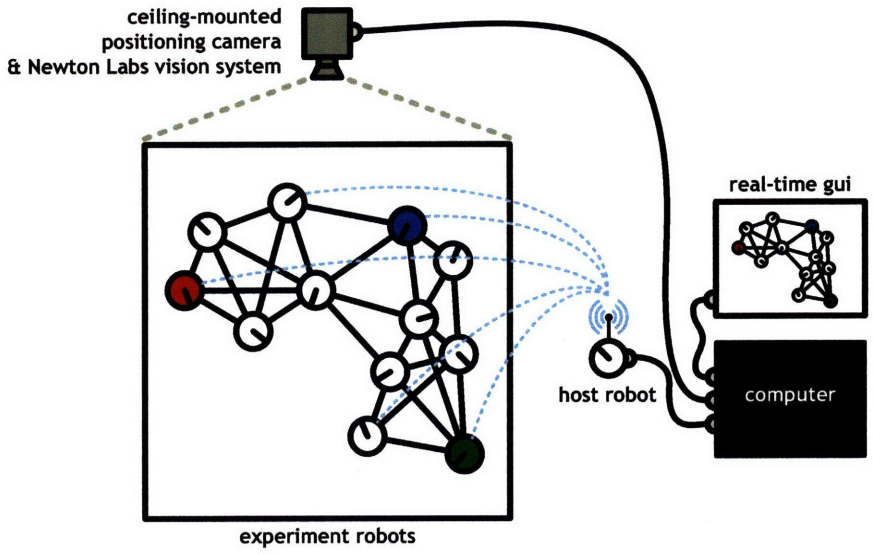
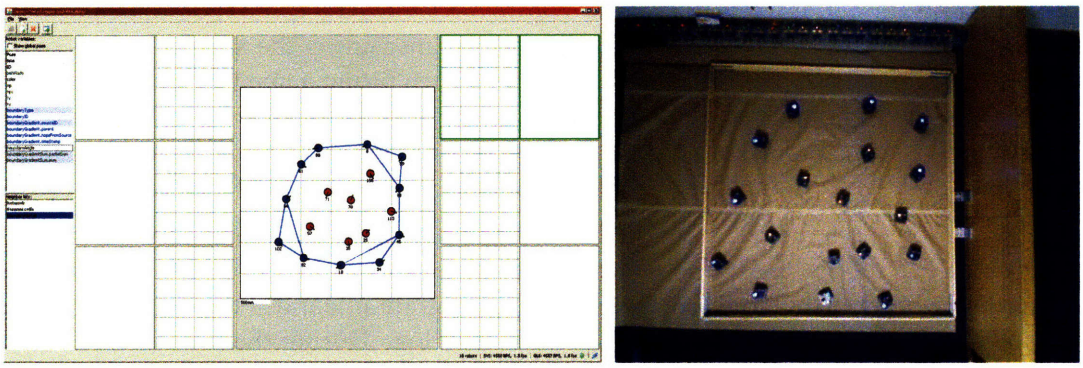


Figure 3-6: The data collection system consists of four parts. Each robot has a top-mounted infrared emitter that flashes with an individual pattern. The ceiling-mounted camera identifies each robot with this pattern and tracks the positions {x, y} of each robot simultaneously, reporting the results to the computer at 1 hz. The host robot uses its radio to query each experiment robot and reports their logged variables to the computer. The computer unifies both data streams to present a real-time graphical display to the user and log data for future analysis.



(a) The data collection software

(b) View from the ceiling-mounted camera.

Figure 3-7: **a.** The data logging software. **b.** There are two overhead cameras, one for recording video, and the other for logging the positions of the robots.

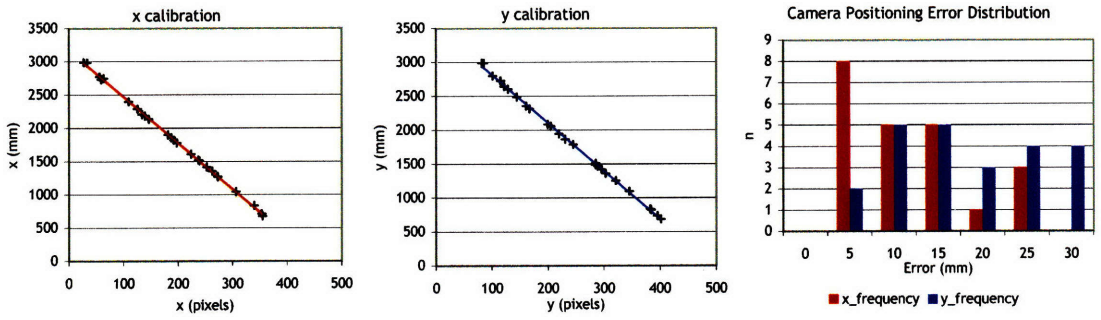


Figure 3-8: The calibration data and error histograms for the vision-based localization system. The x-axis shows measured error and the y-axis shows the bin population. The mean error over x and y axes is 15.4 mm. The system occasionally produces errors in excess of 30 mm, but refused to do so when we were testing it. These larger errors are very intermittent, occurring about once every 500 frames. These errors are very drastic shifts in position lasting for only one frame, and are easily filtered out.

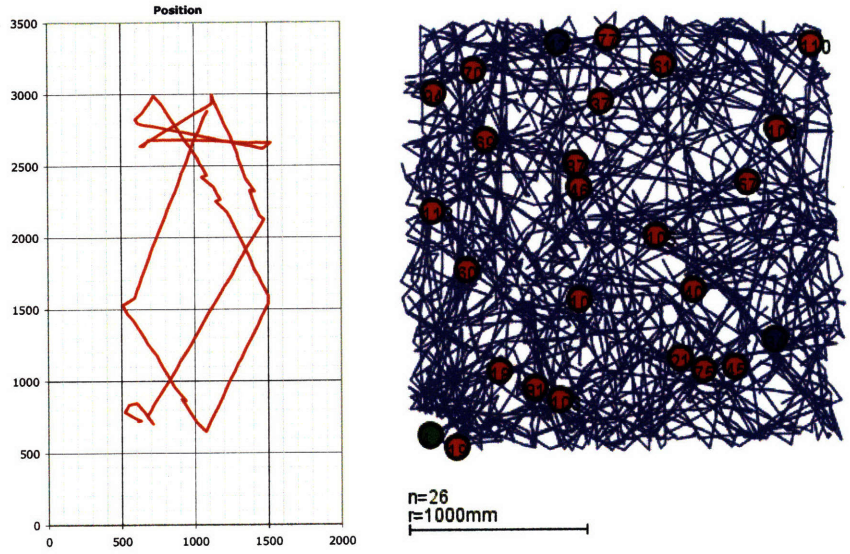
3.4.2 Data Collection Infrastructure

A diagram of the data collection infrastructure is shown in Figure 3-6. There are four main components: the ceiling mounted infrared(IR) camera, the host robot, the computer, and data logging software.

Ground truth is determined by a vision-based localization system. The system was developed by Newton Labs [62], and tracks the positions, $\{x, y\}$, of all of the robots simultaneously. Each robot has an IR emitter on the top circuit board. This emitter blinks with an encoded pattern that allows the vision system to decode 10 bits of data per second per robot. Each robot uses its robot ID as the unique pattern, which allows us to identify its individual position in the recorded data. Figure 3-8 shows the calibration data and position error histograms. The system was calibrated by measuring the x and y values of the physical positions of 25 robots in the environment, then fitting a line to the reported pixel values from the camera. The linearity is quite good, and the mean error is 15.4 mm. The system reports the positions of all the robots once per second. Because of the 1 hz update rate, we limit the maximum speed of the robots in all experiments to 80 mm/s.

A host robot uses the radios to query all of the robots for their telemetry during execution. With 25 robots, the frame rate of this system is also 1 hz. This data is sent to the data logger program shown in Figure 3-7a. This program integrates the camera positioning data with the telemetry to produce a composite log. During the process, the logger also combines the local neighbor pose estimates with the global positions to produce an estimate for each robot's global heading.

The system is effective, but the limited field of view of the camera limits the workspace size to about 3 m $\times$ 3 m. Also, the limited memory on the robots prevents logs from being stored locally, so the rate of data collection is limited by the radio bandwidth. This, in turn, limits the temporal precision of the collected data. This can present a problem when recording fast-moving events, such as broadcast tree formation.



(a) The path from a single robot running BUMPREFLECT

(b) The paths from many robots running BUMPREFLECT

Figure 3-9: All of the algorithms were tested while the robots were executing the same background motion behavior in order to test the robustness of the algorithms to rapidly changing configurations. The BUMPREFLECT behavior drives the robots in a straight line until they contact an obstacle. The robots rotate twice the measured angle of incidence, "reflecting" off of the obstacle and back into the interior of the workspace. The behavior is effective at causing robots to change neighbors frequently, and keeping the density of robots roughly uniform throughout the environment.

3.4.3 Experimental Protocol

Each algorithm was evaluated under the same experimental conditions. Each algorithm is run on 25-35 robots moving around the environment. With the exception of the navigation algorithm described in Section 5.6, all of the algorithms use the BUMPREFLECT motion behavior. This behavior moves the robots in a straight lines until they contact an obstacle, then use the bump sensors to estimate the angle of incidence to "reflect" the robot back into the environment. Figure 3-9 shows the path of a single robot running this behavior, and the combined paths of many robots. The behavior is good at keeping robots dispersed throughout the environment, and the constant mobility changes the robot's neighbors frequently. This makes it a good test to characterize how sensitive an algorithm's performance is to changing configurations.

Each experiment tests the algorithm over a wide range of robot speeds and communications parameters shown in Table 3.1. The speed and communication are combined to calculate the robot speed ratio (RSR), which will be introduced in Chapter 4. We tested each algorithm in a static configuration and over a range of robot speed ratios spanning two orders of magnitude; starting from a RSR of 0.005, and doubling until a RSR of 0.640. This range of speeds is most visible when plotted on logarithmic axis, so we round the static configuration up to a RSR of 0.001 to plot on a log scale. Unless otherwise noted, any accuracy result at a RSR of 0.001 is for a static configuration. Each algorithm was tested at

trial	RSR	τ(sec)	s_{robot} (mm/s)	min time (min:sec)
1	0	0.25	0	0:12
2	0.005	0.25	20	0:12
3	0.01	0.25	40	0:12
4	0.02	0.25	80	0:12
5	0.04	0.5	80	0:25
6	0.08	1	80	0:50
7	0.16	2	80	1:40
8	0.32	4	80	3:20
9	0.64	8	80	6:40

Table 3.1: Robot speed ratios used for data collection.

each speed for at least 50 rounds, $10diam(G)$ rounds, or 3 minutes, whichever was longer. Experiments with distinct trials, such as the navigation algorithm from Section 5.6 were executed 10 times. An experiment will typically require about 60 minutes of robot execution time, and take about two hours of actual time to complete, including setup, changing test conditions, swapping robot with dead batteries, and other assorted robot wrangling.

Most plots of algorithm accuracy contain between 20,000-70,000 total data points. The mean values and standard deviation are plotted in the detail plots, along with a random sampling of 100 data points to give an impression of the distribution. The robot speed is measured from the data frames, and this measured value, not the commanded value, is used in the plots. Because successive samples come from successive frames of data, not all of these samples are independent. It can be difficult to decide what an independent sample is in some experiments, so for consistency, we used all the samples for computing statistics. Future work on metrics should consider this problem more carefully.

3.5 Summary

The requirements from the canonical application and practical assumptions drove the development of our multi-robot computational model. This model captures the key features of multi-robot systems, network geometry and group configurations. It also incorporates an abstracted communication system and discrete global execution rounds, which ease analysis. The next chapter will use this model to define four complexity metrics tailored to evaluating the performance of distributed algorithms running on multi-robot systems. Finally, with model and metrics in hand, we can define the dimensionless measure of robot speed that is one of the key contributions of this work.

Chapter 4

Multi-Robot Complexity Metrics and the Robot Speed Ratio

Complexity measures characterize an algorithm's performance based on its use of physical computational resources, traditionally memory space and execution time. Complexity metrics for multi-robot systems must consider a broader range of physical resources, such as robot mobility and network communications. These extra resources can often limit algorithm performance before processor speed or memory size. These extra constraints are robot speed, inter-robot communications bandwidth, the number of robots and their arrangement in physical space. To address this, I propose four additional complexity metrics for characterizing the performance of multi-robot distributed algorithms: physical accuracy, physical running time, communication complexity, and configuration complexity.

The first section of this chapter describes the four multi-robot complexity metrics. *Physical accuracy* is similar to correctness in traditional algorithms, and measures the algorithm's performance in the physical world. For example, the physical accuracy could measure how well the robots achieve a desired physical configuration, or how efficiently they perform a task. The *physical running time* of an algorithm must take into account computation speed, but also the total time for the final physical configuration to be achieved, which will depend on robot speed, among other things. Communication bandwidth is a critical resource in multi-robot systems, and the *communication complexity* of an algorithm must consider the available bandwidth between neighboring robots and the type of messaging required to adapt to changing network topology. Multi-robot systems store algorithm state in the memory of the processors and in the physical configuration of the group. The *configuration complexity* should capture the minimum number of robots required for an algorithm, the amount of information stored in their configuration, and the algorithmic cost of "writing" and "reading" this information.

The second section in this chapter describes five global communications structures of increasing complexity, based on the worst-case propagation distance of a message in the network and the network stability required for correct operation. Some algorithms can run correctly with only one-hop communications with neighboring robots, while others need a global convergecast in order to produce an accurate result.

In the final sections, I define and analyze a platform-independent measure of robot speed. Called the *robot speed ratio*, it is appropriate on mobile systems that rely on multi-hop networks as their primary communication technique. We will use this speed ratio in the remainder of the work to quantify how fast robots are moving relative to their multi-

hop messages, and how this speed affects the accuracy of their algorithms.

4.1 Related Work

There is much work in the literature on measuring the complexity of robots and their tasks. Erdmann [30] develops a model of robotic computation based on understanding how much sensing information is required to solve a particular task. The idea was to understand what sensing was actually needed, and let the problem design the sensor, instead of the other way around. Donald [27], followed by O'Kane and LaValle [84,85], both propose the notion of reductions and dominance in robots and sensors. A robot or sensor dominates another if it can do all the tasks of the former. that is, if it can *simulate* the former. This allows the authors to construct a partial ordering of robots, sensors and the tasks they can solve. In Rus, Donald and Jennings [101], the authors develop four protocols for rotating an object using different kinds of sensory information, then use the formalisms from [27] to prove that the protocols are indeed equivalent.

In general, these approaches strive to answer questions of the sort: "Can this robot solve this problem at all?" or, "Are these two robots equivalent?". In this work, we assume that a given multi-robot system with an appropriate algorithm A is capable of solving the desired problem. These are analogous to computability questions in computer science. We are concerned with answering questions such as: "How well will the system perform?", "How long will it take?", or "How many bits will we need to communicate per second?". These are analogous to complexity metrics in computer science, where the "big O" notion characterizes the performance of an algorithm that is already known to operate correctly. We want to abstract from the actual hardware far enough to compare the performance of algorithm A across disparate platforms, but we also want to get the answers to our questions in units of meters, seconds, and bits.

Kannan and Parker [53] develop a method for characterizing a system's ability to respond to faults to produce a fault-tolerant architecture. They implement this system on two mobile robots and empirically demonstrate their technique is more robust than an established method, the Casual Model Method. The metric they use is borrowed from multi-processor systems to characterize if a given a fault-tolerance system is ineffective, or represents a good balance of performance and fault-tolerance. This metric returns a value normalized between $[0, 1]$ to facilitate comparisons across disparate systems. We use this same idea of a normalized metric in the definition of physical accuracy.

Although most of the above work was developed for multi-robot systems, it is unclear how it will scale to populations of hundreds of robots. Spears et. al. [42,106–108] develop a crystallization algorithm that runs in real life on a seven robots and in simulation on hundreds, but the main relevance to this work is their definition of error. They define the global quality of a crystalline formation as the average over a local geometric relationship between two pairs of agents. This quality metric has a well-defined ideal performance, and a worst-case performance that can be computed in expectation for a random configuration of agents. Many of our algorithms have a similar accuracy structure, with a well-defined ideal case, and a worst case that occurs when the configuration is uniformly random.

In the previous chapter, we made the assertion that communication and mobility are related in multi-robot systems. Robots that move must communicate to discover new neighbors, and as robots move faster, they will need to communicate more frequently. Guibas et al. [6,75] describes *Kinetic Data Structures* that are computational geometry algorithms that

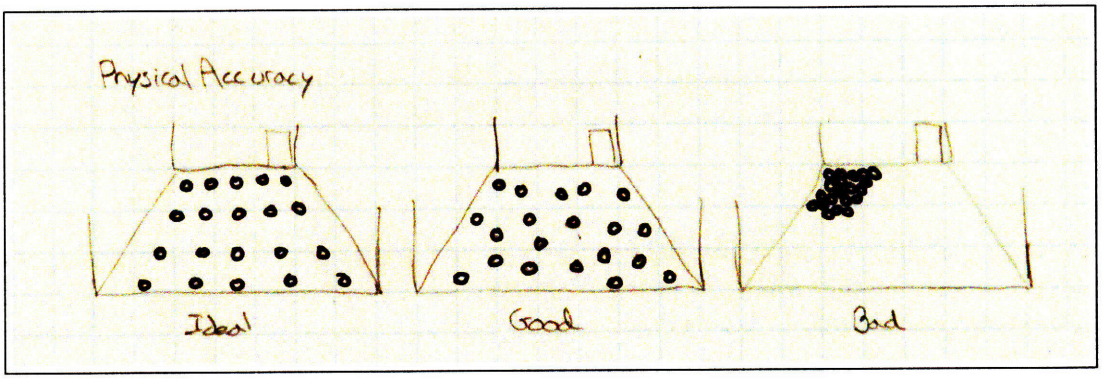
operate on a dynamic geometric graph in the plane. They are designed to update themselves under changing network topology, with provable upper bounds on metrics such as the spanning ratio, and good run-time performance in terms of messages required to keep the structures up-to-date in dynamic configurations. While this work does not address the problems we consider, it is a solid theoretical foundation of minimum communication requirements in dynamic network topologies.

Our definition of the propagation speed of messages in the configuration network is based directly on the work of Klienrock and Silvester [55] on determining the node degree that produces the maximum throughput in an ad-hoc wireless network. Their throughput analysis required them to describe the relationship between the *expected distance per hop*, and the average node degree. Given a start and end node, the expected distance per hop is the distance a message travels through the network projected onto the straight line between the start and end node. This concept is explained in more detail in Section 4.4.1.

4.2 Complexity Metrics

In this section, we define four complexity metrics for characterizing the performance of multi-robot distributed algorithms: physical accuracy, physical running time, communication complexity, and configuration complexity.

4.2.1 Physical Accuracy



The physical accuracy, $\mathcal{A}_A$, of a multi-robot algorithm A, is a real-valued scalar, $0 \leq \mathcal{A}_A \leq 1$. The accuracy of a multi-robot algorithm is analogous to the correctness of a traditional algorithm, but allows for a range of values to indicate "how correct" the algorithm output is. Because a physical configuration contains real-valued quantities, a binary metric could allow small sensing or positioning errors to cause an entire configuration to be classified as incorrect. For example, a dispersion algorithm might specify that all inter-robot separation distances, d_{ab}, fall within a range $d_{min} \leq d_{ab} \leq d_{max}$. A binary metric for correctness would return FALSE if any edge length falls outside this range, while a real-valued accuracy metric can compute the average accuracy and return that instead. The user can then decide what accuracy threshold is sufficient for their application.

In general, the physical accuracy of algorithm A is the ratio of a norm of the achieved configuration to a canonical desired configuration, or the achieved performance to an ideal performance:

$$\mathcal{A}_{\mathrm{A}} = \frac{|\text{actual performance}|}{|\text{best possible performance}|}.$$

The norm used in the above equation is based on the specification of the algorithm and the intended application, and can be any real-valued function such that $|\text{configuration}| \geq 0$. There is no clear way to automate the process of deriving an accuracy expression from an algorithm specification, as the accuracy of an algorithm can depend on many subjective factors, most importantly the user's intended use of the algorithm in the particular application. The best approach is to make an accuracy expression a required part of the specification of any multi-robot algorithm. In this work, the accuracy expression is given for each algorithm.

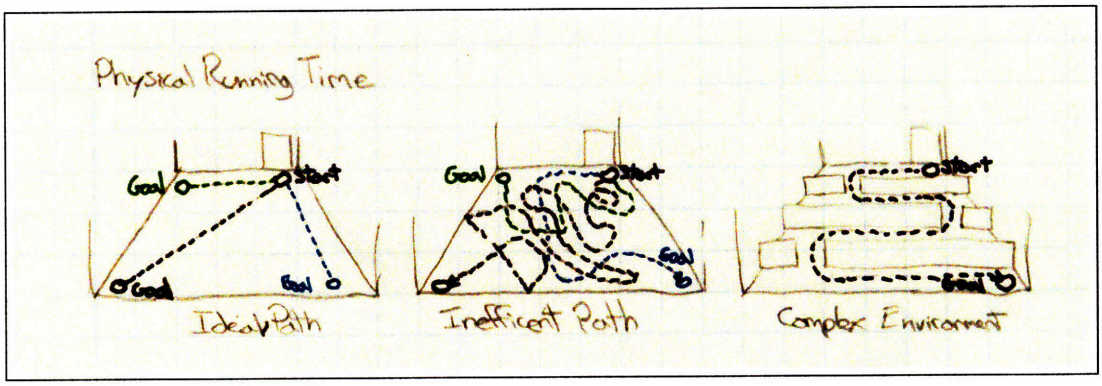
In some cases, the most appropriate accuracy metric might not be an instantaneous measure of configuration accuracy at a single point in time, but instead a measure of efficiency or efficacy over a well-defined interval. For example, the navigation algorithm discussed in Section 5.6 is designed to guide a robot from anywhere in the network to the physical location of any other robot in the network. The accuracy of this algorithm is defined as:

$$\mathcal{A}_{\text{navigation}} \equiv \frac{\text{distance of the actual path traveled by the robot}}{\text{the shortest possible path that could have been traveled by the robot}},$$

where a value of 1 would correspond to perfect path efficiency, and a value of 0 would indicate the navigating robot traveled an infinite path, that is, was not able to reach the desired destination robot. Intermediate values indicate a loss of efficiency in the path, but still with a correct final outcome. Labeling all of these intermediate outcomes with an accuracy of one would obscure differences in efficiency that might be relevant to the application.

The accuracy requires that the underlying distributed algorithm be correct in the traditional sense, and that execution of this algorithm evolves the configuration through a series of valid configurations to a desired configuration. During experiments, we want to measure the steady-state accuracy of an algorithm. In the previous chapter, we assumed that all multi-robot algorithms are self-stabilizing, and are continually able to respond to disturbances. But sensor errors and other system noise can make it difficult to determine when the robots have reached their final configuration. In this work, we average accuracy values over time scales much longer than this system noise, and always more than $10diam(G)$ rounds. If this is insufficient and we need to define a more specific final configuration, we will explicitly specify what steady-state accuracy means.

4.2.2 Physical Running Time



There are two separate performance factors that contribute to the physical running time, $\mathcal{T}_A$, of a multi-robot algorithm A. The first is the computational complexity of the algorithm in the traditional sense. The second is the time it takes for the robots to achieve the final configuration, which is dependant upon many things, including the physical speed of the robots, their path efficiency, the complexity of the environment, and the number of communications rounds the algorithm requires. Any of these two factors can dominate the system to produce a lower bound on total running time.

Computation complexity in a multi-robot system is the standard definition: The number of operations an algorithm requires to produce a new result. The difference in our model is that if a computation completes within a single round, the processor will remain idle until the next round. Therefore, the only situation where the computation complexity affects the physical running time is when the computations cannot complete during a single round. Often, the computation is based on sensor readings, neighbor positions and inter-robot communications, and the computation performed is relatively simple. We can assume that processing time for sensors is $O(1)$ rounds w.r.t. our key system parameters, albeit with a large constant for sensors like cameras. The time to process neighbor messages grows as $\Omega(m)$, where m is the number of neighbors. In our system, the maximum number of neighbors is small enough so that even algorithms with poor scaling properties, e.g. $O(m^3)$, can be computed within one round.

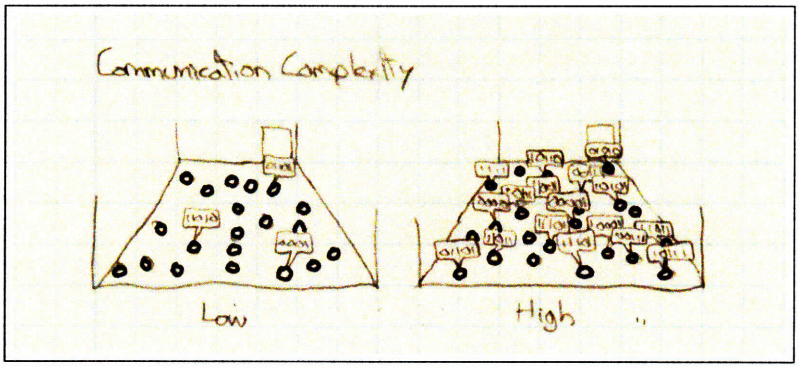
Many multi-robot algorithms rely on multi-hop broadcast communication. These messages require $O(diam(G))$ rounds to propagate throughout the network. Usually, the time for a message to propagate the diameter of the network is small compared to the time for a robot to cover the same distance. But this is not always the case, and the the minimum number of rounds of computation must be considered with evaluating the physical running time of an algorithm.

For algorithms that require mobility, the speed of the robots can also be the limiting factor on physical running time. The minimum time for any robot to complete its task in the algorithm can be computed from the robot's maximum speed and the distance along the shortest traversable path from the robot's initial position to its final position. The maximum over all the shortest times of the robots is the minimum running time for the entire algorithm. This minimum time will vary greatly depending on the particular algorithm and the environmental constraints. For example, we would expect the physical running time of a dispersion algorithm that starts with all the robots at a single point and achieves a uniform density in an open space to grow as $\sqrt{n}$. This is because adding more robots will increase the area of the final configuration by $O(\sqrt{n})$, and the maximum distance traveled

by a single robot will be the diameter of the final configuration, which grows as $O(\sqrt{n})$. But this same dispersion algorithm running in a constrained indoor environment with narrow corridors might have a much longer running time, as the $diam(C_{\text{final}})$ might be a serpentine path through the environment that is much longer than the radius of the circular blob produced in an unconstrained environment.

In general, we would expect that algorithm performance in a large, open environments to be constrained by the number of robots, while robots operating in a complex, maze-like environment would have a running time that depends more on the complexity of the environment. In this work, we focus only on the physical accuracy, not the physical running time. We leave the development of this area as future work.

4.2.3 Communication Complexity



Since each robot has a finite amount of local communications bandwidth to send and receive messages, the most appropriate unit of measure for communications complexity, C_{A}, of an algorithm A, is *bits per robot per round*. Because we assume a shared fixed round duration, τ, we can compute the total bandwidth (bps) required by algorithm A on a single robot with m neighbors:

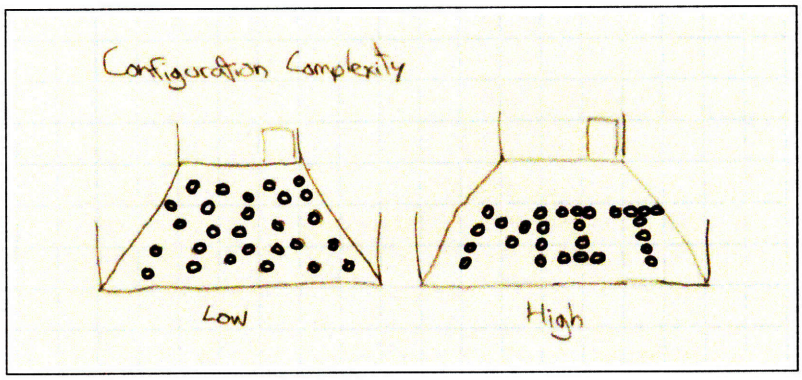
$$bps_A = \frac{mC_{\text{A}}}{\tau}.$$

This is more useful than measuring the total number of bits required to complete the algorithm, because distributed algorithms on multi-robot systems must run continuously in order to allow the system to respond to changes in the population, sensor reading, or the environment. Because of this, the total number of bits used is $bps_A\mathcal{T}_{\text{A}}$. If we assume a constant message size. For some algorithms it is possible to compute a lower bound on the number of messages that need to be sent to reach the first stable configuration. However, the actual number of messages used in an execution will depend on the physical running time, and the number of errors and disturbances the algorithm must cope with.

Communications can quickly become the critical resource in the system. Algorithms that use less inter-robot bandwidth are preferred, and algorithms in which communications complexity grows as $O(n)$ or greater are infeasible on all but the smallest populations. The period of each round, τ, allows a direct trade-off between the bandwidth used and the latency of discovering new neighbors. Assuming the robots are communicating constantly and using all the available capacity on the channel, shorter rounds will allow robots to discover and update neighbors more rapidly, allowing for a faster physical speed during execution. Also, many multi-robot applications have running times that grow as

$O(diam(G))$ rounds, so it is doubly desirable to have each round be as short as possible. We will define a measure of robot speed below that captures some of these relationships, and explore the relationship between physical speed and communication complexity in the rest of this work.

4.2.4 Configuration Complexity



As a multi-robot system executes an application, it stores algorithm state not only in the memory of the processors, but also in the intermediate physical configurations the group produces as it progresses towards the goal. There are many different questions to consider about this configuration complexity:

- How much information is stored in the pose of each individual robot?
- How much information is stored in the entire configuration?
- What is the minimum physical configuration information required for task completion?
- What is the algorithmic cost of storing and retrieving this information?
- What is the minimum number of robots required to complete a particular task?
- What are the costs of losing the information stored in a given robot?

We do not address these questions here, and leave this fascinating area of research for future work.

4.3 Global Communication Structure

The metrics above do not completely describe the complexity of a given algorithm. Different algorithms propagate information throughout the network in different ways. In Section 3.3 we described three different techniques for network communications, one-hop, broadcast tree, and convergecast. The *global communications structure* of an algorithm describes the most complex way that algorithm uses multi-hop communication. The algorithms in this work use five types of global communications structure:

No Communication: Algorithm accuracy requires no inter-robot communication, therefore algorithm accuracy does not depend on robot speed.

One-Hop Communication: Algorithm accuracy only requires information that has traveled a maximum of one hop; *i.e.* information generated on or measured from neighboring robots. For example, robot a can use information robot b can measure directly, such as robot b's sensor values, to compute a local one-hop average light value. However, robot a cannot use robot b's average light sensor value, because it could have been computed using information from robot c, violating the one-hop limitation. The algorithm output will be accurate after all robots receive information from their neighbors. In a static configuration with no message loss, this is one round.

Broadcast Tree Communication: Algorithm accuracy requires that all robots receive a message and that propagation of this message constructs a spanning tree structure, T, in the network. Tree construction algorithms are described in Section 3.3.2. The algorithm output will be accurate after all robots have received the message, and by doing so, computed their parent in the broadcast tree. In a static configuration with no message loss, this will take $depth(T)$ rounds. However, the depth of the tree will depend on the particular robot that is the root. It is often convenient to note that $depth(T) \leq 2diam(G)$, and the running time of any algorithm that uses this global communication structure is $O(diam(G))$.

Broadcast Tree + Convergecast Communication: Algorithm accuracy requires broadcast tree construction + convergecast aggregation to propagate the results back to the root. These operations happen simultaneously. While the broadcast tree, T, is being constructed, there is a flow of convergecast results toward the root. However, correct results do not reach the root until the tree is complete, and the quantities from the deepest robot have propagated back to the root. In a static configuration with no message loss, the root will receive correct convergecast results in $2 \cdot depth(T)$ rounds.

Broadcast Tree + Convergecast + Rebroadcast Communication: Algorithm accuracy requires broadcast tree construction, convergecast communication, then another broadcast to distribute the results throughout the network. Like the previous structure, these operations happen simultaneously. While the broadcast tree, T, is being constructed, there is a constant flow of convergecast results toward the root, and a constant flow of rebroadcast results towards the leaves. In a static configuration with no message loss, all robots will receive correct rebroadcast results in $3 \cdot depth(T)$ rounds.

In all algorithms that use any communication, we expect the physical accuracy to decrease as robot speed increases. However, the type of global communication structure an algorithm uses will change the way algorithm accuracy degrades. We would expect that the more complex communication structures would degrade at earlier speeds, because the messages have further to travel, and because the convergecast requires a stable broadcast tree to propagate results towards the root.

4.4 The Robot Speed Ratio

In our assumptions from Chapter 3, we assert that the robot's mobility requires that they discover and update their neighbors at regular intervals, and we adopt periodic neighbor updates to support this and produce a manageable computational model. As robots move

faster, the network will be changing faster, and the frequency of the neighbor updates must increase to maintain the communication network. Since communications bandwidth is limited, there is a critical robot speed where the update frequency can no longer be increased. If the robots move faster than this speed, the network can no longer be properly maintained, and the performance of any algorithm that relies on network communication will degrade. Since each robot also relies on the network connectivity to produce its local network geometry, performance of configuration control algorithms will degrade as well.

In this section, we use the assumptions and computational model from the previous chapter to define a dimensionless measure of robot speed that encapsulates the relationship between robot speed and communications bandwidth. We use this speed measure in the rest of the work to characterize how an algorithm's accuracy degrades as the speed increases.

We desire a measure of robot speed that:

1. Captures the relationship between communications usage, communications range, round duration, and robot speed;
2. Can be calculated from the system parameters shown in Table 4.1, to enable it to be used as a design tool for new systems; and
3. Is not based on a particular system architecture, and can be used for algorithm performance comparisons across disparate hardware platforms.

Our approach is to define the robot's speed as a fraction of the maximum message propagation speed, s_{message}. Given the actual robot speed, s_{robot}, the *robot speed ratio*, or RSR, is

$$RSR = \frac{s_{\text{robot}}}{s_{\text{message}}}. \tag{4.1}$$

The maximum message speed is not the speed of the message in the physical medium, but instead the propagation speed of the message through the network. This includes buffering delays, retransmissions, and channel capacity limits. Any robot moving with away from a broadcast tree root with a RSR > 1, that is, faster than the maximum message propagation speed, will not be able to communicate with the robots behind it. This effectively disconnects the network, making it impossible for any distributed algorithm to run correctly. This is a worst-case analysis, as motion towards the root, or tangential to the root, or in parallel with root motion, would not be subject to this limit. But this requires knowing the exact direction of motion for all robots and the exact root position, making the metric difficult to apply to a general-purpose execution, and requiring much more information to make it useful at design time.

4.4.1 Message Speed Analysis

To determine the robot speed ratio, we need to be able to calculate and measure s_{robot} and s_{message}. For land-based systems, we will assume the actual speed of the robots does not deviate very far from the commanded speed. This is a reasonable assumption, unless there is an extraordinary amount of wheel slippage. For non-terrestrial systems, we will assume that the robot speed can be specified and measured with sufficient accuracy for our calculations. Note the robot speed is something the algorithm designer needs to know,

n	Total number of robots.
E_{area}	Area of the environment. (m^2)
ρ	Average density of robots in the environment. (#/m^2)
s_{robot}	Maximum speed of the robots executing the algorithm. (m/s)
r	Communications range. (m)
τ	Round duration. (s)
B	Maximum communications bandwidth per robot. (bits/s)
A	The algorithm being used.
$\mathcal{C}_{\text{A}}$	Communications complexity of algorithm A. (bits/round)

Table 4.1: System parameters. These are constants that we can determine about our system without executing software. Note that some of them are related under certain assumptions. For example, $\rho = \frac{n}{E_{\text{area}}}$ if robot density is assumed to be uniform.

but not the robot itself. This is similar to the use of the global pose in the definition of configuration; the robot does not have access to its pose, but it is required to define the configuration.

We turn our attention to computing the message propagation speed. In a multi-hop communication network, the speed of message propagation can be measured by noting the minimum round-trip time (RTT) for a message to travel from any node a to another node b, and the Euclidean distance between the two:

$$s_{\text{message}} = \frac{\text{distance}_{ab}}{RTT_{ab}/2}.$$

While the above expression allows us to empirically measure the message speed, it does not allow us to calculate it from the fundamental system parameters in Table 4.1. In order to support this calculation, we use the fixed communication round duration τ to estimate the expected latency per hop, t_{hop}, and the information about the configuration graph, G, estimate the average distance traveled per hop, d_{hop}. We can then express the message speed as

$$s_{\text{message}} = \frac{d_{hop}}{t_{hop}}.$$

This expression will allow us to calculate the expected message speed for a given system from system parameters.

The first step is to calculate the average distance traveled per hop in the graph G. We provide an intuitive discussion of this calculation here; see the paper by Klienrock and Silvester [55] for a careful analytic treatment.

A *spanning subgraph* of a graph is a subgraph that has some edges removed, but is still connected on the same set of vertices. We can view the network graph, G, of a configuration as a spanning subgraph of the fully connected graph of that configuration, G^c, with edges where $|e| > r$ removed. This is called an r-disc graph, or a unit-disk graph if $r = 1$. Figure 4-1 illustrates a unit-disk spanning subgraph of a fully connected graph. We define d^c_{ab} as the length of the shortest path from node a to b using the edges in G^c, and d_{ab} as the length of the shortest path from node a to b using the edges in G. The *spanning ratio*, $t \in \mathbb{R}$, is the maximum *stretch* in path length between d_{ab} and d^c_{ab} over all pairs a, b

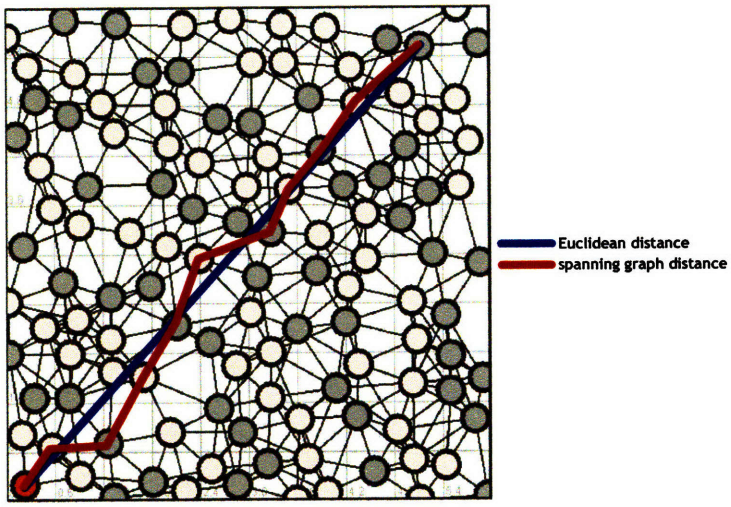


Figure 4-1: The unit-disk spanning subgraph, G', of a fully connected graph, G. The edges shown are those of G', the edges of G are omitted for clarity. The root is the red circle in the lower left of the figure. Robots that are an odd number of hops from the root are shaded light red, those that are an even number of hops are shaded in grey. The blue path is the Euclidean path; the corresponding edge would exist in a fully connected graph. The red line is the path through the spanning subgraph. Note that the length of the spanning path is always greater or equal to the Euclidean distance.

$$t = \max\left(\frac{d_{ab}}{d^c_{ab}}\right).$$

Note that $d_{ab} \geq d^c_{ab}$ for any a, b, therefore $t \geq 1$. We are concerned with the aggregate properties of the graph rather than the worst-case properties of individual paths, so we define the the average spanning ratio for all pairs of nodes a, b as

$$k = \text{mean}\left(\frac{d_{ab}}{d^c_{ab}}\right),$$
$$d_{ab} \approx k d^c_{ab}.$$

We want to use the spanning ratio to estimate the distance a message travels per communication hop along the straight line path between two nodes. Let h be the number of hops along a particular path through the network between two nodes. We can express the minimum number of hops along the crooked path ab in G:

$$h_{min} = \left\lceil \frac{d_{ab}}{r} \right\rceil \approx \frac{k d^c_{ab}}{r}.$$

We have allowed fractional hops (or assumed a very large number of hops that allows us to safely remove the ceiling operators) to clarify the math, and replaced d_{ab} with the previous equation. The path through the fully connected graph, d^c_{ab}, is also the Euclidean distance between these two nodes. The message covers this distance using the number of hops computed above, h_{min}, therefore

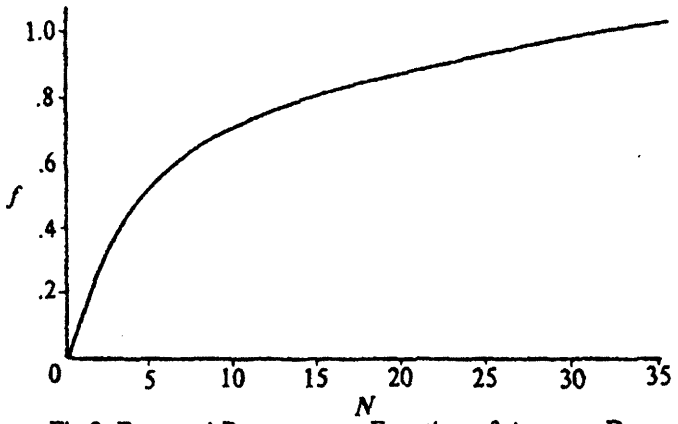


Fig 2. Expected Progress as a Function of Average Degree

Figure 4-2: As the configuration's density increases, there will be shorter paths through the spanning subgraph between any two robots. Assuming a communications radius of 1, we can calculate the expected progress per hop, d_{hop}, as a function of average degree of a vertex in the configuration graph, *i.e.*, the average number of neighbors of a robot (From Klienrock and Sylvester [55]).

$$d_{hop} = \frac{d_{ab}^{c}}{h_{min}}.$$

Substituting the previous equation for h_{min} gives us

$$d_{hop} = \frac{r}{k}. \tag{4.2}$$

This makes sense intuitively. A network with a smaller spanning ratio supports paths that are closer in distance to the Euclidean distance, *i.e.*, it is more likely that a robot will have a neighbor that is better aligned with the straight-line path of the message, and the expected progress per hop increases. For $r = 1$, the expected progress per hop is the reciprocal of the average spanning ratio, $d_{hop} = \frac{1}{k}$. A graph plotting d_{hop} for $r = 1$ for increasing average node degree is shown in Figure 4-2.

The average spanning ratio k, becomes a fundamental parameter in our model, and we can estimate it *a priori* if we assume a uniform density of robots in the environment, ρ. The average degree per node, m_{avg}, can be estimated with

$$m_{avg} = \rho\pi r^2 - 1 \approx \rho\pi r^2.$$

For the configurations tested in this work, $m_{avg} \approx 10$, yielding $k \approx 1.4$.

The next step is to compute t_{hop}, the expected latency at each hop. We assume that while each robot uses the same round duration, τ, they all have different offsets, picked uniformly at random from $0 \leq \tau_{\text{offset}} < \tau$. Therefore, the most delay between receipt and retransmit of a message by any robot is $\tau - \epsilon$, where ϵ is an arbitrary small constant bounded away from 0. The minimum is σ, where σ is some small processing time. Rounding ϵ and σ both to 0, we can compute the expected time delay at each node as

$$t_{hop} = \frac{\tau}{2},$$

and now the message speed is:

$$s_{\text{message}} = \frac{d_{hop}}{t_{hop}} = \frac{2r}{k\tau}. \tag{4.3}$$

The round duration is listed as a system parameter, but it cannot be set arbitrarily small: it is bounded below by the communication requirements of the algorithm. Given the maximum communication bandwidth of each robot, B, and an assumption on the maximum number of neighbors for each robot, m_{max}, we can compute the minimum round time for a given algorithm's (A) communication usage measured in bits/round, C_{A}:

$$\tau_{min} = \frac{C_{\text{A}}\, m_{max}}{B},$$

so that

$$t_{hop} = \frac{C_{\text{A}}\, m_{max}}{2B}, \tag{4.4}$$

Note that this assumes that the physical communications medium is a shared resource and is half-duplex. This assumption is true for most of the radio and optical communications hardware used in a practical multi-robot system.

We can now compute the message speed in terms of Eq. 4.2 and Eq. 4.4:

$$s_{\text{message}} = \frac{d_{hop}}{t_{hop}} = \frac{2rB}{k\, m_{max}\, C_{\text{A}}}. \tag{4.5}$$

This expression makes sense when we vary the system parameters from Table 4.1. Increasing r, the communication range, allows the message to travel further with each hop. Increasing B, the bandwidth, decreasing C_{A}, the algorithm's required bits per round, or reducing m_{max} the number of neighbors, all allow the round to be shorter, which increases message speed. A smaller spanning ratio, k, gives the message a straighter, more efficient path. Sharing local communications resources with fewer neighbors allows the round length to be smaller. Unfortunately, m_{max} is not one of our initial parameters, but we can estimate an adequate substitute, m_{avg}.

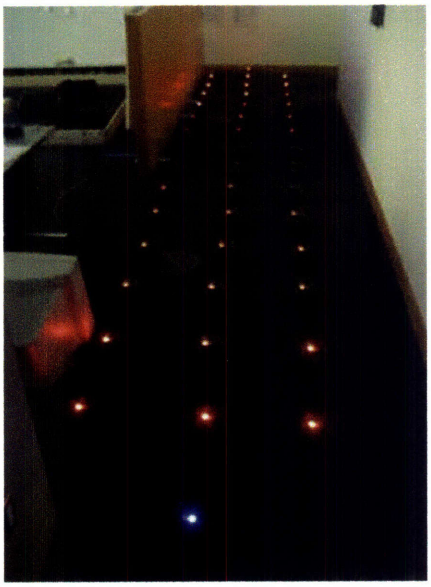
Assuming a uniform distribution of robots, we can use the environment area, E_{area}, and the total number of robots in the system, n, to compute the density of robots, ρ:

$$\rho = \frac{n}{E_{\text{area}}}.$$

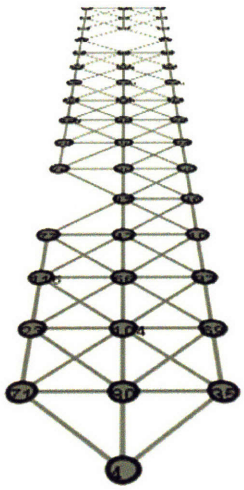
Neglecting boundary effects, we use the expression for m_{avg} from above, which gives us

$$s_{\text{message}} = \frac{2B}{k\pi r\rho\, C_{\text{A}}} = \frac{2\, B\, E_{\text{area}}}{k\pi r n\, C_{\text{A}}}. \tag{4.6}$$

This expression is not nearly as intuitive, but it is in terms of the fundamental system parameters that we specified above. Note that because r and m_{avg} are now related with our uniform density assumption, r appears in the denominator of these expressions. This occurs because increasing the radius will add neighbors proportional to the communications area, $O(r^2)$, while only increasing the message speed proportional to $O(r)$.



(a) Physical arrangement of robots



(b) Communication network from experiment

Figure 4-3: **a.** Picture of the message speed test configuration. 49 robots were used in this experiment. **b.** The robot's network from the telemetry data. Note that two robots are missing in the data, as they depleted their batteries during the experiment.

Finally, we can express the robot speed, s_{robot} as a fraction of the message speed from Equations 4.3, 4.5, or 4.5 to produce three equivalent expressions for the *robot speed ratio*, or RSR:

$$RSR = \frac{s_{\text{robot}} k \tau}{2r}, \tag{4.7}$$

$$RSR = \frac{s_{\text{robot}}\ k\ m_{max}\ C_{\text{A}}}{2rB}, \tag{4.8}$$

$$RSR = \frac{s_{\text{robot}}\ k\pi r n\ C_{\text{A}}}{2\ B\ E_{\text{area}}}. \tag{4.9}$$

4.4.2 Message Speed Experimental Results

Measurement of the message speed required a different experimental setup, as the square environment in which we conduct most of our experiments was too small to produce a network large enough for effective measurement. Unfortunately, moving out of the workspace meant the robots were out of view of the overhead positioning camera, so each robot's location had to be measured manually. To facilitate the position measurement process, the robots were arranged in the grid pattern shown in Figure 4-3. Because the robots' one-hop communications system is optical, this configuration caused neighboring robots to occlude each other, which effectively limited their communication range to the diagonal of the grid spacing, d, $r = d\sqrt{2}$. We can determine k from the average neighbor count and Figure 4-2. Neglecting edge effects, the average neighbor count for this arrangement is 8, giving us an expected progress of $d_{hop} = 0.65$, and a spanning ratio, $k = \frac{1}{d} = 1.54$. The

round duration, τ, was 0.250 seconds. Given these constants, we can compute ideal the message speed from Equation 4.3:

$$s_{\text{message}} = \frac{2r}{k\tau} = 3.36\text{m/s}$$

In this case, we can take advantages of the regularities in this configuration to improve our calculation by noting that the network geometry constrains any hop away from the root to advance one grid length, d, away from the root. If we let $r = d$ and $k = 1$, we get a $s_{\text{message}} = 3.66$ m/s.

The speed measured from a video recording was $s_{\text{message}} = 3.64$ m/s. Both of the calculations are within 10% of the measured value.

4.4.3 Robot Speed Ratio Case Study: A Tale of Two Robots

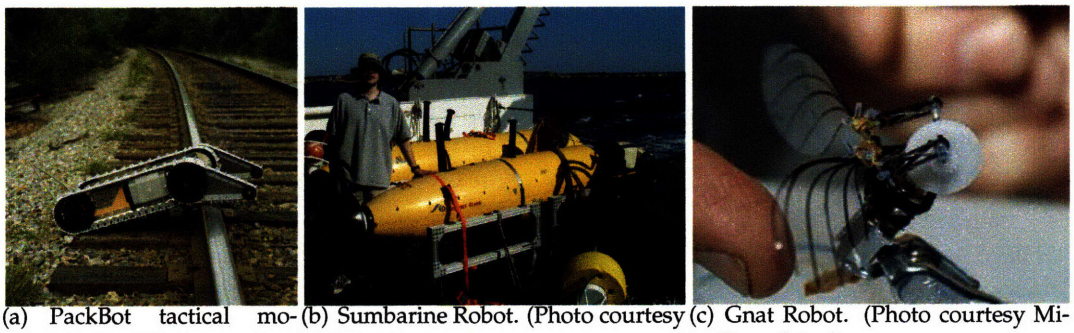
A comparison of the parameters and subsequent RSR's for a standard research robot, the B-21 from Real World Interfaces (no longer in production), and the SwarmBots [52] used in these experiments is shown in Figure 4-4. Using reasonable estimates of the system parameters of the B-21 and SwarmBot robots yields RSR's that are five orders of magnitude apart! The cause of this discrepancy is the range of the communications systems and the bandwidth available to each individual robot. A seasoned practitioner could quibble with some of these estimates, but not enough to close the gap by more than one order of magnitude. It would still be the case that messages propagate at radically different physical speeds on networks formed by these robots, but the robots themselves move at around the same speed.

In systems where the RSR is very low, communications propagate much faster than the robots move. This allows proofs of communication algorithms to assume that the configuration is static as messages propagate, and proofs of mobility algorithms can assume that messages are updated instantaneously. As the RSR increases, these assumptions no longer hold. A reasonable hypothesis is that the physical accuracy of a given algorithm would degrade as the RSR increases. Some algorithms might even have a *critical RSR*, a speed beyond which it can no longer function properly. The goal for the rest of this work is to measure how the RSR affects the accuracy of a variety of algorithms with different global communications structures.

There are many environments and systems where the RSR might be high. In the case of the SwarmBots, it is the combination of a small communication range and a low bandwidth communications system. Figure 4-5 shows three other example systems that might have a high RSR. A PackBot in an open environment with long-range, high-bandwidth radio communications might have a very low RSR, but the same robot operating in an urban environment or a building would have much smaller effective radio range, and a higher RSR. The communications range of an unmanned underwater vehicle can be very large, measured in kilometers, but the bandwidth limits on underwater communications are extremely low, on the order of hundreds of bytes/sec. Systems like this might even operate at RSR greater than one, and physically deliver messages between neighboring robots instead of transmitting them. Microaerial vehicles (gnat robots) may present the greatest challenge, as they will probably have short-range, low-bandwidth communications, coupled with fairly high robot speeds.

Parameter	B-21 Robot	SwarmBot
k	0.7	
m_{max}	16 neighbors	
C_A	3,072 bytes/round	
s_{robot}	0.1 m/s	
r	100 m	1 m
bandwidth	6,592 kbytes/s	15 kbytes/s
$s_{message}$	39,237 m/s	0.91 m/s
RSR	**0.0000025**	**0.11**

Figure 4-4: The B-21 Research robot (left) and the SwarmBot (right). The differences between the system parameters of these two robots produce very different robot speed ratios.



(a) PackBot tactical mobile robot. (Photo courtesy iRobot).

(b) Sumbarine Robot. (Photo courtesy Ed Olsen).

(c) Gnat Robot. (Photo courtesy MicroPropulsion).

Figure 4-5: Different robots in different environments can have different robot speed ratios. **a.** A PackBot tactical mobile robot operating in a cave will have very limited communications range **b.** A UUV can have very large communications range (kilometers), but very low bandwidth (only 100's of bytes/sec) **c.** A micro-UAV might have very short-range and low-bandwidth communications, while still having a fairly high robot speed..

4.4.4 Robot Speed Ratio Trade-Offs

Recall the expression of the RSR from Equation 4.8:

$$RSR = \frac{k\, m_{max}\, s_{\text{robot}}\, C_{\text{A}}}{2rB}$$

The maximum RSR that has any chance of correct execution is 1, but I will present empirical results in upcoming chapters indicating that a RSR of around 0.02 yields good accuracy on our experimental system. It is reasonable to assume that for a given system, there is a maximum RSR that produces acceptable algorithm accuracy. Given any fixed RSR, the system designer can trade off other system parameters to achieve the same algorithm accuracy.

For example, the user can directly trade bandwidth for physical robot speed. Using an algorithm with a lower communications complexity (C_{A}) would also allow the robots to move faster. A run-time density management algorithm could cause robots to disperse from areas where the neighbor count $m > m_{max}$, which could help maintain the bandwidth utilization below the maximum bandwidth of the system.

4.5 Summary

The proposed metrics are useful for quantifying the performance of multi-robot algorithms as they execute on physical systems. We focus on the physical accuracy and communications complexity in this work. The robot speed ratio couples physical speed with communications, allowing direct trade-offs between these two quantities, as well as other system parameters. The rest of the thesis is spent evaluating the physical accuracy of multi-robot distributed algorithms as a function of the RSR. The next chapter tests standard preexisting algorithms in this new context.

Chapter 5

Communication and Navigation Algorithms

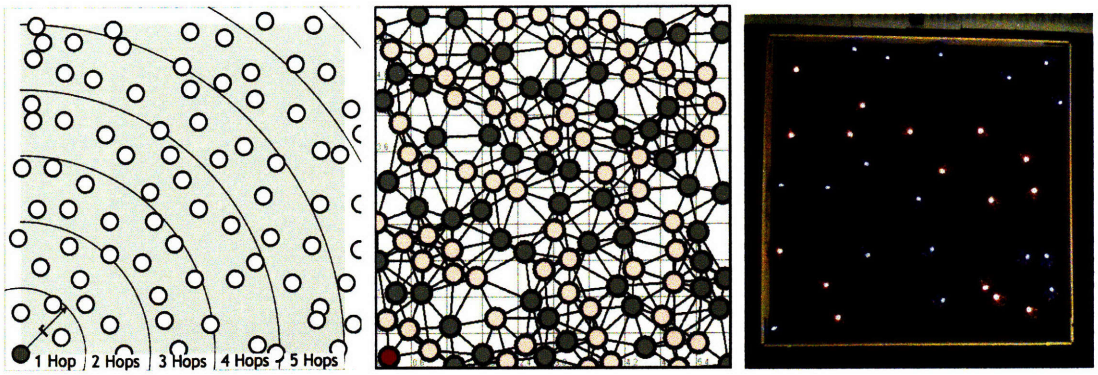
5.1 Introduction

This section characterizes the performance of fundamental preexisting algorithms for communication and navigation for networked robots at a variety of robot speed ratios. The four algorithms considered include two algorithms to characterize the global geometry of broadcast tree construction, one algorithm that uses broadcast trees as an computational routing structure to aggregate global quantities onto a single robot, and a navigation algorithm that uses the broadcast tree as a physical routing structure to guide robots towards the root. The broadcast tree algorithm is described briefly in Section 3.3.2, and in detail in reference [73]. In multi-robot systems, broadcast trees are used for more than communications: their geometric properties are used in constructing coordinate systems, routing messages, and providing navigation aids for mobile robots [11, 79]. In this chapter, we focus on the geometric properties of broadcast trees and how they degrade as the robot speed ratio increases.

The broadcast tree propagates outward from the root as robots communicate with their neighbors. Each time the message is relayed, or *hops*, from robot to robot, it covers some physical distance. Figure 5-1a illustrates the distribution of hops for an ideal propagation with the root robot at the lower left-hand corner of the plot. Ideally, each message would travel the maximum communication distance, r, from each robot at each hop, forming a series of non-overlapping concentric rings, with each successive ring containing robots that are one hop further from the source than those in the previous ring.

However, because robot density is not infinite, there are not always robots at the locations required for this ideal propagation. This produces the crooked paths of the unit-disk spanning subgraph shown in Figure 5-1b. In this illustration, the root robot is drawn in red, robots that are an odd number of hops from the root are drawn in light red, and those that are an even number of hops are drawn in grey. The rings are no longer circular, and the average distance covered per hop along a straight line from the root is reduced to $d_{hop} = \frac{r}{k}$, from Equation 4.2.

Figure 5-1c shows an image of real robots running the broadcast tree construction algorithm in a static configuration (RSR = 0). In a static configuration, the implementation closely matches the predictions. This is because the robots for our experiments use an infrared communication system, which has a carefully calibrated and uniform commu-



(a) Ideal hop distribution (b) Simulated hop distribution (c) Hop distribution on robots

Figure 5-1: Broadcast tree construction. **a.** The ideal distribution of communication hops would be a series of concentric circles around the root. **b.** A simulation of a hops distribution from a root robot in the lower-left. Robots with an odd and even number of hops are drawn in light red and grey respectively. **c.** Picture of the robots indicating their hops from the root of a broadcast tree. The root is the single blue robot located in the lower left corner of the workspace. Robots that are located an even number of hops from the root are flashing their blue light, those located an odd number of hops are flashing their red light.

nications range that is well-modeled by the unit-disk graph. In a network using radio communications, the communications range is a complex function of antenna design and location in the environment, and is not well-modeled by a unit-disk graph. However, since we assume that mobile robots are able to localize their neighbors and estimate their local network geometry, we could use this geometric information to produce a unit-disk overlay onto a radio network.

In this chapter, we study the physical accuracy of the broadcast tree by examining how closely the geometrical relationships in moving configurations match those in static configurations. We look at two algorithms that use the broadcast tree to estimate the magnitude and direction to the root robot, and how their accuracy is affected by dynamic configurations at varying robot speed ratios. We then consider two algorithms that use the broadcast tree for routing; one for routing communications messages towards the root, and another for guiding robots towards the root.

5.2 Related Work

Broadcast tree construction is a standard form of communication in ad-hoc networks and sensor networks [50]. This type of communications is essentially a distributed version of breadth-first search, and generates a spanning tree rooted at a distinguished root node in the network. This technique is a practical way to propagate a global broadcast throughout the network; the messages propagate rapidly, without cycles, and decay in an orderly fashion. In this section, we focus on the geometric properties of the tree that is constructed, and how these properties degrade at high RSRs.

Convergecast is a common technique for accumulating global data onto a single robot [68, 69]. Robots in the network use the broadcast tree as a routing structure to relay convergecast messages towards the root. The difference from standard routing is that con-

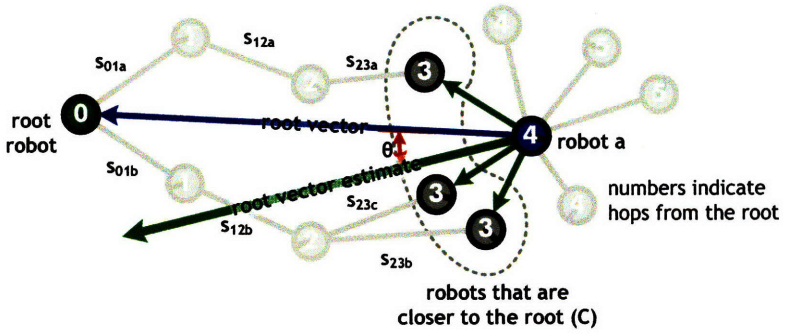


Figure 5-2: The broadcast tree can be used by any robot in the network to estimate the vector to the root. Robot a is four hops from the root. It has three neighbors that are closer than it is, one of them is its parent in the tree. Robot a can compute an estimate of the direction to the root by averaging the directions to its three neighbors that are closer to the root. The distance to the root can be estimated by summing the length of the edges the broadcast tree message has traveled to reach robot a.

vergecast messages received by a single robot are aggregated, and this result is then propagated towards the root. This eliminates the need to route each message individually, significantly reducing the communications requirements from $O(n)$ bits/round to $O(1)$ bits/round. However, the algorithm requires a stable routing tree to correctly propagate convergecast messages back towards the root.

There is much work describing navigation algorithms that uses the broadcast tree for physical routing [11, 63]. These algorithms guide a robot towards the root in a series of steps, by moving towards the parent in the broadcast tree. A requirement of these algorithms is a strong correlation between a robot's estimate of its distance to the root and its actual distance to the root.

In this chapter, we implement variants of these popular algorithms, define accuracy metrics for them, and evaluate their performance at a variety of robot speed ratios. The goal is to quantify the relationship between the broadcast tree data structure and the global network geometry, with an emphasis on supporting navigation algorithms.

5.3 Broadcast Tree Root Vector Distance

We define the *root vector* as the vector from any robot to the root of the tree. The root vector provides long-range geometric information suitable for navigation and other applications. We use two seperate algorithms to estimate the direction and magnitude of this vector. This section describes an algorithm to estimate the root vector magnitude, and the next section describes an algorithm for estimating the root vector direction.

5.3.1 Problem Definition

We desire an algorithm to measure the distance to the root of a broadcast communication tree. We define the root vector, $\overrightarrow{R}$, as the vector from any robot to the root of the tree. We define the root vector estimate, $\overrightarrow{R_{est}}$, as the estimate of the root vector made by an

running time (rounds)	$O(diam(G))$
computation per round	$O(m)$
global communications structure	broadcast tree
$\mathcal{C}_{\text{RootVectorMagnitude}}$	$O(1)$
$\mathcal{A}_{\text{RootVectorMagnitude}}$	corrcoef$(\|\overrightarrow{R}\|, \|\overrightarrow{R_{est}}\|)$

Table 5.1: Properties of the root vector magnitude algorithm.

individual robot. Figure 5-2 illustrates the root vector, robot a's estimate of it, and the information the two algorithms in this section use to compute the estimate.

Individual robots use their local network geometry to estimate the direction and magnitude of the vector. Estimating the root vector requires a broadcast tree. Every robot is a member of this tree, and therefore has at least one neighbor that is closer[1] to (fewer hops from) the root than it is. We examine a particular robot a that is not the root of the tree. Robot a has $k \geq 1$ neighbors that are closer to the root than it is. We gather these closer neighbors into the set $\mathbf{P}_a$, and use the information that a knows about these closer neighbors to calculate the root vector estimate $\overrightarrow{R_{est}}$. Robot a knows the pose of each robot $r_i \in \mathbf{P}_a$ from its local network geometry. The position vector p_i of each neighbor r_i is the first two components of its pose: $\overrightarrow{p_i} = \left[\text{pose}_i.x, \text{pose}_i.y\right]^T$.

5.3.2 Algorithm

To estimate the root vector magnitude, $\|\overrightarrow{R_{est}}\|$, we add a public variable, *treeDistance*, to the state of each robot. The value of this variable is available to all neighbors of the robot at the end of each round through the publish/subscribe system. The *treeDistance* variable on robot a is computed by taking the average of the r_i.*treeDistance* from each neighbor $r_i \in \mathbf{P}$ and adding the distance to that neighbor, $\|\overrightarrow{p_i}\|$. The *treeDistance* of the root is 0. The tree distance is calculated on each other robot with the following expression:

$$a.\textit{treeDistance} = \frac{1}{k}\sum_{i=0}^{k}\left(r_i.\textit{treeDistance} + \|\overrightarrow{p_i}\|\right).$$

The magnitude of the root vector on any robot a is the current value of a.*treeDistance*:

$$\|\overrightarrow{r_{est}}\| = \textit{treeDistance}.$$

5.3.3 Analysis

The root vector magnitude algorithm runs concurrently with the broadcast tree, and its results are calculated as soon as a robot receives a broadcast tree message. Therefore the total running time is $O(diam(G))$ rounds, and it uses $O(m)$ computation per round, because $k \leq m$, where m is the number of neighbors. The global communications structure is broadcast tree, and the communication complexity, $\mathcal{C}_{\text{RootVectorMagnitude}}$, is $O(1)$, as the

[1] I'm not calling this set the "parents" because a robot only has one parent in the broadcast tree. The appropriate familial designation would be $\mathbf{P}_a$ = parent $\bigcup$ aunts $\bigcup$ uncles

only information that needs to be shared between neighbors is their hops from the root and the public variable *treeDistance*.

Accuracy Metric

We want the estimated root vector magnitude to be well correlated with the actual distance from the root. We use the correlation coefficient computed between the actual and estimated root vector magnitudes as the accuracy metric:

$$\mathcal{A}_{\text{RootVectorMagnitude}} = \text{corrcoef}(\|\vec{R}\|, \|\overrightarrow{R_{est}}\|).$$

The correlation coefficient returns a result in the range of $[0, 1]$, which is exactly what we desire for the accuracy matric. We compute the algorithm's accuracy without compensating for k, the spanning ratio, as a local measurement of density would introduce errors from edge effects, and a more global estimate would require more complexity and communications. This means that we can never expect to achieve an accuracy of 1 because the spanning ratio in the network makes the measured paths longer than the actual paths. In practice, this path-length error is moderate, but tolerable, usually less than 20%, and a rough estimate of the average robot density can be calculated beforehand to compensate if E_{area} and n are known.

Limitations and Errors

The stretch from the spanning ratio isn't the only source of error. Voids in the network will cause a larger stretch in the measured path length, as the messages must route around such obstacles. In an open environment, these are the two main sources of error.

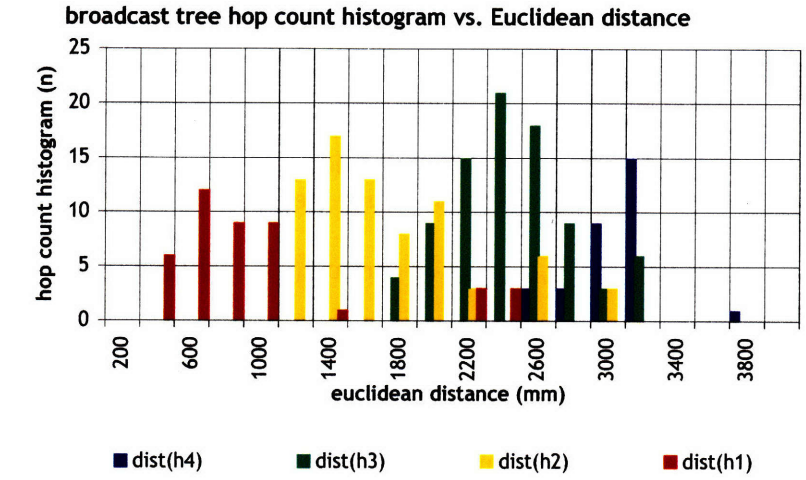
In a convoluted, maze-like environment, the type of local communication and pose estimation system employed will introduce more serious errors. We imagine two types of communication systems, one that computes local network geometry via line-of-sight communication, and one that is able to communicate through obstructions. We will assume that the line-of-sight path between any two robots is also navigable.

The line-of-sight system will produce a distance estimate that reflects the path the broadcast messages traveled, which is also the path the robot should follow to navigate the environment. Producing navigable paths is useful, and we use this property later in this chapter to implement a robust navigation algorithm with good performance. However, the geometric error between the navigable path and the actual distance to the root can be large if the path is convoluted. Therefore, using this distance estimate to produce a coordinate system [79] would produce poor results.

On the other hand, if robots could communicate and estimate local network geometry without interference from obstacles, then the root vector magnitude could return a good estimate of the Euclidean distance to the root, but the path might not be navigable for a robot. In this case, the robots would have to use information from another sensor in order to navigate around obstacles and reach the root.

5.3.4 Experimental Results

The top plot in Figure 5-3 shows a histogram of robots grouped by their hop count from the broadcast tree root for a static configuration. The x-axis is the Euclidean distance of a robot from the root. Each of the four histograms shows the locations of the robots with a



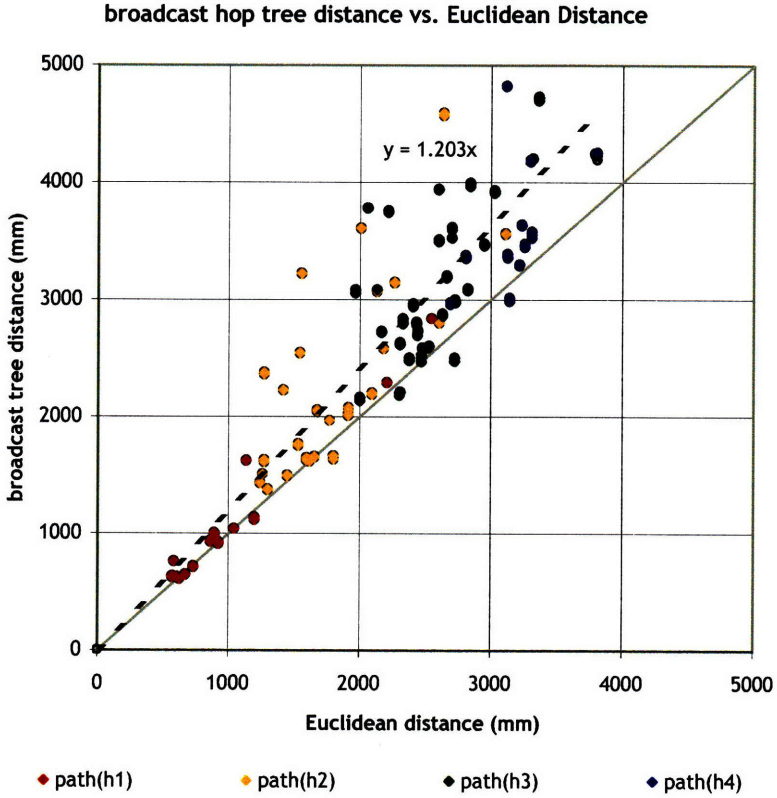


Figure 5-3: **top:** Broadcast tree hop histogram. **bottom** Broadcast tree root vector distance for a static configuration. The distance was overestimated by a factor of 1.18, which is less than the predicted value of 1.43 from the Klienrock equation. This discrepancy is a result of the small hops in the network and the communication range being larger than 1.0 meters

particular number of hops. For example, the red histogram are 1-hop robots, and they are mostly located near the root. The green histogram shows 3-hop robots, and they are further from the root. For a uniform density of robots, the population of histograms further from the root will be larger, because the area of the ring where these histograms are likely to exist is larger. With a communications range of 1 m in our 2.44 m $\times$ 2.44 m square workspace, the area for the 4-hop robots is only a fraction of the total area where these robots could be in an unconstrained environment. This explains why the size 4-hop histogram does not continue to increase, but instead shrinks in size. This same effect can be seen in the picture in Figure 5-1c.

The bottom plot in Figure 5-3 displays the broadcast tree distance vs. the Euclidean distance from the root for a static configuration of robots. The Euclidean distance was calculated by measuring the distance from a given robot to the root from the overhead camera. Since the path through the network is always longer than the Euclidean distance, all of the points should lie above the line $y = x$. The few that are below the line are from local pose estimation errors, where a robot underestimated the distance to its parent in the tree. The least-squares fit through the data gives a stretch of 1.2, which is close to our predicted spanning ratio of 1.4. Some of the discrepancy can be attributed to the robots' actual communications range being larger than 1 meters, which admits more robots as neighbors, and reduces the spanning ratio. Also, this is a small network with a large percentage of one-hop links, which have error only from the network geometry localization, not from the path stretch.

Figure 5-4 shows broadcast tree root vector distance estimates from the robots plotted at six different robot speed ratios. The distance estimate starts as a correlated, but noisy, line at a RSR of 0.003 which looks very similar to the static configuration shown in Figure 5-3. As the RSR increases, the correlation between the robots distance estimate and the actual distance diminishes, and eventually turns into a nearly uniform distribution at the highest RSR of 0.6. The intuition is that robots moving at a high RSR can move a very large distance in each round of computation, carrying their previous hop counts and distance estimates with them as they travel. In turn, this affects their neighbors in future rounds, further reducing the correlation between the root distance estimates and actual distance from the root.

Plotting the accuracy of the root vector magnitude vs the robot speed ratio, we observe a steady decrease in accuracy. Note that we use the robots magnitude estimate, uncorrected with knowledge of the spanning ratio. This means that the robots can never achieve an accuracy of 1, because the path through the network will always be somewhat crooked. The accuracy in the static configurations was 0.90. This accuracy decayed to less than 0.30 at a RSR of 0.64. This supports our intuition.

5.4 Broadcast Tree Root Direction

5.4.1 Problem Definition

We desire an algorithm that can enable any robot in the network to estimate the direction towards the root robot. This has many uses, including producing coordinate systems and navigation.

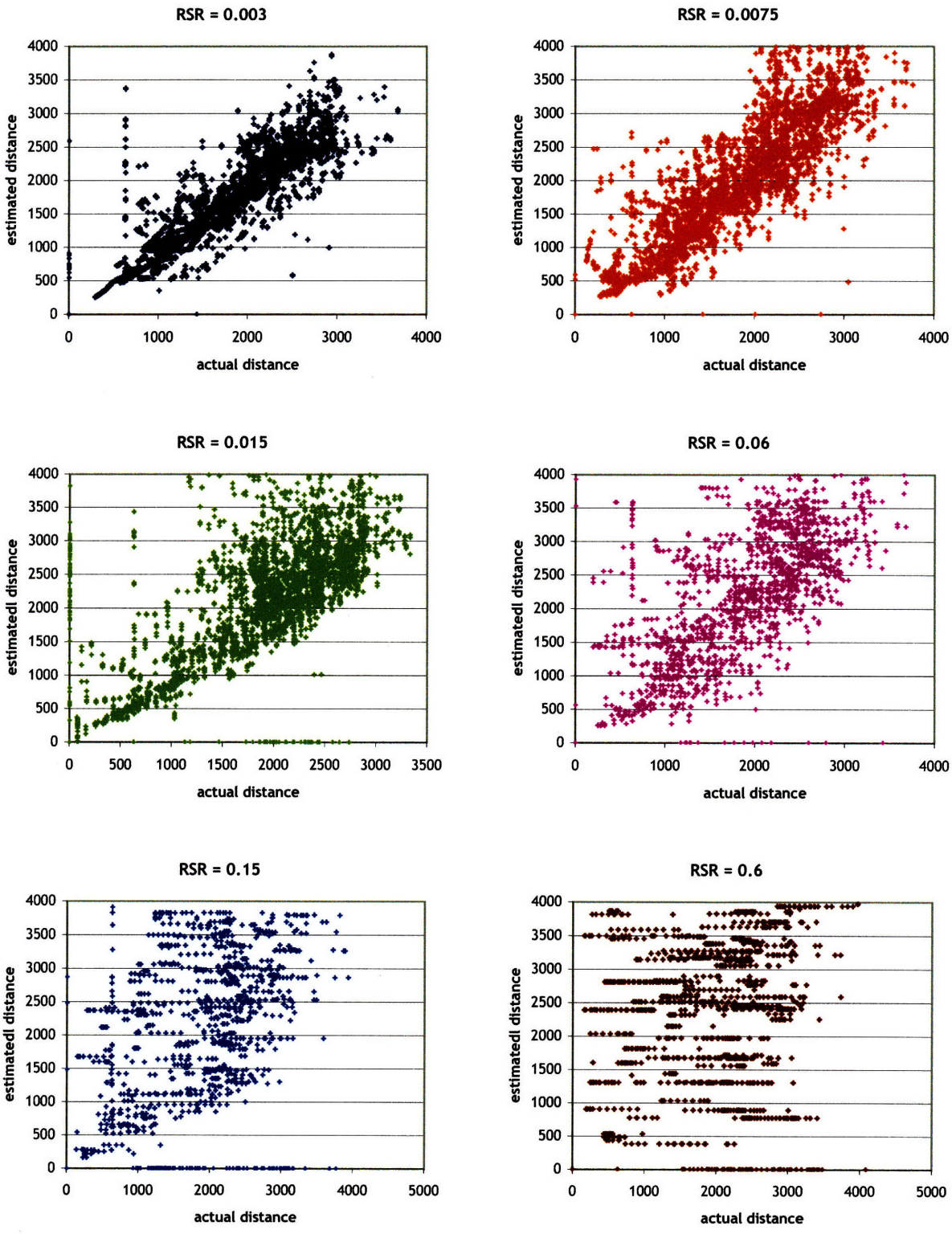


Figure 5-4: Broadcast tree root vector distance distributions for dynamic configurations. The root vector distance estimate deteriorates from a noisy line at low RSR to a uniform distribution at high RSR. This makes sense as the robots move faster, the broadcast tree cannot re-form rapidly enough to produce new distance estimates. At a RSR of 0.6, the positions are very poorly correlated to the root vector magnitude estimate.

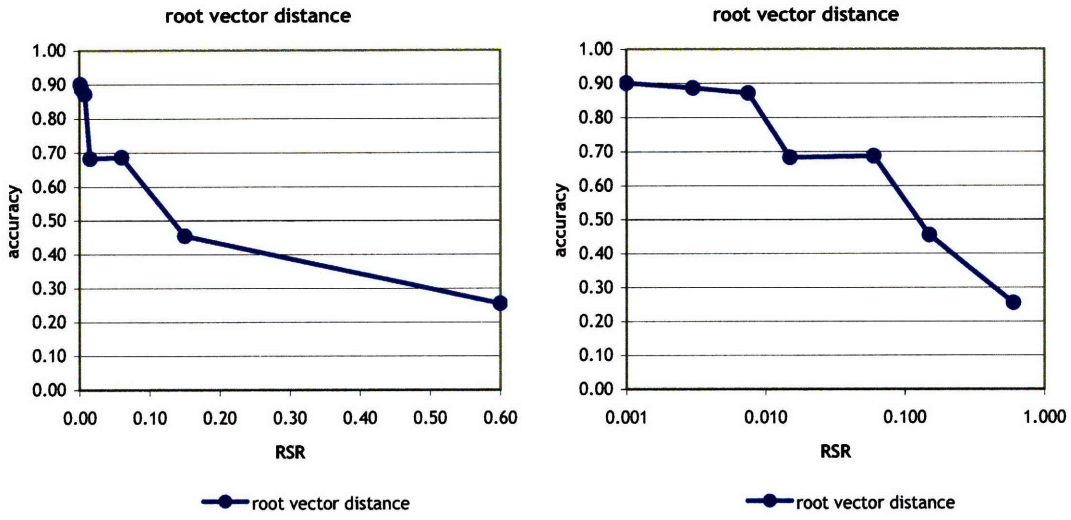


Figure 5-5: The root vector distance accuracy decreases at higher RSRs. The accuracy metric is the correlation coefficient between the actual distance to the root and the measured distance to the root. (The same data is shown on both plots. left: linear x-axis, right: log x-axis).

5.4.2 Algorithm

This root vector direction algorithm uses similar information to the root vector magnitude algorithm. Refer again to Figure 5-2. Again, we examine a particular robot a that is not the root of the tree and collect a's k closer neighbors into the set $\mathbf{P}_a$. The position vector p_i of each neighbor r_i is given by: $\overrightarrow{p_i} = \left[\text{pose}_i.x, \text{pose}_i.y\right]^T$.

The direction of the root vector direction estimate, $\angle\overrightarrow{R_{est}}$, in robot a's coordinate frame is calculated by computing the average direction to all neighboring robots in $\mathbf{P}_a$:

$$\angle\overrightarrow{R_{est}} = \text{arctan2}(\sum_{i=0}^{k} \overrightarrow{p_i})$$

This direction is calculated in robot a's coordinate frame. This is a very naive algorithm. We can produce a better estimate by incorporating the direction estimates from $\mathbf{P}_a$ as well. From a practical point of view, the accuracy of the simple algorithm is sufficiently high (Static accuracy ≈ 0.9) that the development of a more sophisticated version was not needed.

5.4.3 Analysis

As with the magnitude algorithm, the root vector direction algorithm runs concurrently with broadcast tree construction, and its results are calculated as soon as a robot receives a broadcast tree message. The total running time is $O(diam(G))$ rounds, and $O(m)$ computation per round. The global communications structure is broadcast tree, and the communication complexity, $C_{\text{RootVectorDirection}}$, is $O(1)$, as the only information that needs to be

running time (rounds)	$O(1)$
computation per round	$O(m)$
global communications structure	broadcast tree communications
$\mathcal{A}_{\text{RootVectorDirection}}$	$\frac{\pi - \lvert\theta_{error}\rvert}{\pi}$
$\mathcal{C}_{\text{RootVectorDirection}}$	$O(1)$

Table 5.2: Properties of the root vector direction algorithm.

shared between neighbors is their hops on the broadcast tree.

As with the root vector magnitude estimate, internal voids can produce different kinds of errors depending on the type of communication used. If two robots that can communicate with each other can also navigate to each other, then the root vector direction estimate will be aligned with the path the broadcast messages traveled, which is also a navigable path the robot could follow. However, the error of the direction estimate will be entirely dependent on the particular local environment and the arrangement of the obstructions. This could render the direction estimate useless for an application that requires geometric information about the true vector to the root.

Similarly, if robots can communicate and estimate local network geometry without interference from obstacles, this this vector will estimate the true vector to the root, but the path might not be navigable for a robot.

5.4.4 Accuracy Metric

We choose to evaluate the accuracy of the root vector direction estimate in the context of navigation. We desire an accuracy metric that assigns a value of 1 to the estimate if is directly aligned with the navigable path, and a value of 0 if it is facing the wrong direction. Furthermore, we want to be able to predict what kind of accuracy we will have in a highly mobile configuration with very infrequent communication, *i.e.* high RSR. To this end, we define the physical accuracy metric for the root vector direction, $\mathcal{A}_{\text{RootVectorDirection}}$, to be:

$$\theta_{error} = \angle\overrightarrow{R_{est}} - \angle\vec{R}$$

$$\text{normalize } \theta_{error} \text{ to lie within } -\pi \text{ and } \pi$$

$$\mathcal{A}_{\text{RootVectorDirection}} = \frac{\pi - \lvert\theta_{error}\rvert}{\pi}$$

With this metric, we would expect a group of robots with high RSR to produce a uniform distribution of accuracies. An alternative metric is produced by using the dot product between the actual and estimated vectors:

$$e = \overrightarrow{R_{est}} \cdot \vec{R}$$

$$\mathcal{A}_{\text{RootVectorDirection}}{}' = \begin{cases} \text{e} & \text{if } e > 0 \\ 0 & \text{otherwise} \end{cases}$$

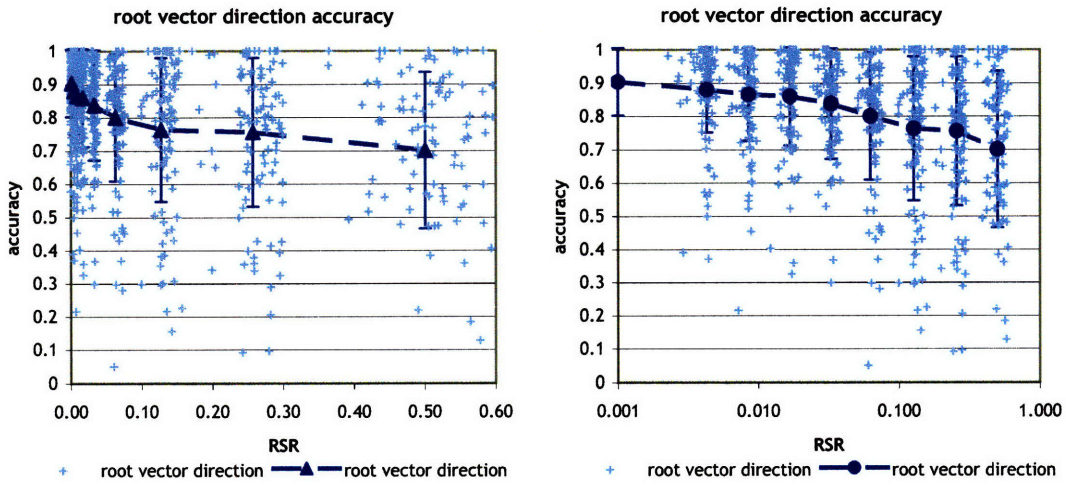


Figure 5-6: The root vector direction accuracy decreases at higher RSRs. We would have expected the value to approach 0.5 at high RSR. The slightly higher value could be explained by robots near the source still receiving messages from the source, and not being able to carry them to an arbitrary location in the environment. (The same data is shown on both plots. left: linear x-axis, right: log x-axis).

However, the non-linear *cos* term in the dot product makes the determination of the accuracy in the random high-RSR configuration more difficult to calculate. But these metrics are essentially equivalent, and we use the first accuracy metric in this work.

5.4.5 Experimental Results

Figure 5-6 displays the value of $\mathcal{A}_{\text{RootVectorDirection}}$ over a range of robot speed ratios. At high RSR, we would expect the mean of the accuracy to approach 0.5, and the distribution to become more uniform. The mean of the accuracy degraded to 0.7, and the variance increased, but did not become uniform. One source of error is due to the fact that even at a RSR of 0.640, the robots only travel a max of 0.64 meters per round. Robots who are within communications range of the root when they receive a message from it will only travel a limited distance from the root, and not be located in an arbitrary position at the end of the next round. Assuming this new one-hop robot is heading away from the root, we can compute the expected distance it travels in one round:

$$\begin{aligned} E[\text{distance traveled}] = {} & E[\text{distance from root when message received}] \\ & + E[\text{distance per round after message}] \end{aligned}$$

At a RSR of 0.64, with $r = 1.0$ m, $\tau = 8$ s, $s_{\text{robot}} = 80$ mm/sec, $E[\text{distance traveled}]$ between any robot and the stationary root evaluates to $0.5 + 0.32 = 0.82$ meters over one round. Therefore, we would expect that the distribution of one-hop robots is not uniformly random, and that there are more one-hop robots closer to the root. This effect will continue

to all other hop counts: there will always be a slight concentration of one-hop robots within some extended radius of the root, and two-hop robots within some extended radius of the one-hop robots, and so on. This slight hop distribution gradient will be apparent in any configuration where the root robot is stationary and the other robots cannot assume an arbitrary position in the environment in one round. An algorithm with a mobile root would perhaps approach a more random distribution, especially in an environment where the root robot can move to any location during a single round. In general, determining the accuracy limit for high RSR configurations depends on the specific algorithm, the metric, and the environment, because robots only travel a limited distance per round.

5.5 Convergecast Summation

The convergecast summation algorithm described in Section 3.3.3 is useful to compute global quantities, such as the the population size. The global boundary classification algorithm in Section 6.6 uses a convergecast summation to compute the exterior angle of the boundary polygon. The tree-based dynamic task assignment algorithm in Section 7.5.5 uses a convergecast summation to compute an estimate of the current task distribution. Convergecast is a more complicated global communication structure, as it requires the broadcast tree to be stable enough to route messages back to the root.

5.5.1 Problem Definition

We desire an algorithm that can compute global quantities across the configuration. The algorithm cannot require more than $O(1)$ bits/round of communication bandwidth.

5.5.2 Algorithm

The algorithm requires a broadcast tree to operate. Every robot in the network is a member of this tree, and therefore has a parent in the tree. The algorithm also requires that each robot in the tree know which of its neighbors are its children in the tree. Our implementation accomplishes this by adding a public variable, *parentID*, to each robot. When a robot selects a broadcast tree message to store (see Section 3.3.2), the sender of this message becomes its parent, and it stores the ID of this robot in its *parentID* variable. This allows each robot to identify its children by querying the public shared variables of all of its neighbors, looking for any neighbor with a *parentID* that is equal to its own ID.

The algorithm adds a second public variable, *partialSum*, to the state of each robot. The algorithm sums the quantity q by relaying the values of *partialSum* up the broadcast tree towards the root. We examine a particular robot a. Robot a has $k \geq 0$ children in the broadcast tree. We collect these children into the set $\mathbf{C}_a$. We refer to each child of a as $c_i \in \mathbf{C}_a$. Robot a computes its partial sum in each round by adding its quantity, q_a, to the partial sum of each of its children, $c_i.partialSum$:

$$a.partialSum = q_a + \sum_{i=0}^{k} (c_i.partialSum)$$

The partial sum computed on robot a is stored in its public variable, $a.partialSum$, making it accessible to all of its neighbors.

running time (rounds)	$O(diam(G))$
computation per round	$O(m)$
global communications structure	broadcast tree + convergecast
$\mathcal{A}_{\text{convergecast}}$	$\frac{q_{\text{measured}}}{q_{\text{actual}}}$
$\mathcal{C}_{\text{convergecast}}$	$O(1)$

Table 5.3: Properties of the convergecast algorithm.

By this process, each robot in the tree computes the partial sum of q over the subtree rooted at itself. The subtree rooted at the root robot is the entire tree, and the partial sum computed by the root robot is the total sum of q over the entire configuration. In our experiment, we choose to compute the total number of robots in the configuration, so each robot lets their value of $q = 1$.

5.5.3 Analysis

The algorithm has a total running time of $2depth(T)$ rounds, where T is the depth of the broadcast routing tree. We approximate this to $O(diam(G))$ rounds. The convergecast algorithm runs concurrently with the broadcast tree construction, but correct results only reach the root robot after $2depth(T)$ rounds after startup or any population change. Since $k \leq m$, there is $O(m)$ computation per round. The global communications structure is convergecast, and the communication complexity, $\mathcal{C}_{\text{convergecast}}$, is $O(1)$.

5.5.4 Physical Accuracy Metric

We desire a physical[2] accuracy that captures both overestimates and underestimates of the actual quantity q. We start by defining the error of the measurement of q:

$$e = \frac{|q_{\text{measured}} - q_{\text{actual}}|}{q_{\text{actual}}}$$

Unfortunately, this error can grow without bound if q_{measured} is much larger than q_{actual}. To handle this, we define the accuracy as a piecewise function to handle large errors:

$$\mathcal{A}_{\text{convergecast}} = \begin{cases} 1 - e & \text{if } e \leq 1 \\ 0 & \text{otherwise} \end{cases}$$

This assigns an accuracy of 0 to any large error, producing a value between $[0, 1]$ for all values of q_{measured} and q_{actual}.

5.5.5 Experimental Results

The convergecast summation algorithm accuracy data is shown in Figure 5-7. The experiment was to count the total number of robots in the configuration. The accuracy degrades quickly as the RSR increases. This is expected, because the algorithm relies on a stable broadcast tree to relay partial sums through its neighbors back towards the root robot.

[2]In this case, the accuracy metric is not really physical. But it does require the broadcast tree to be correctly built.

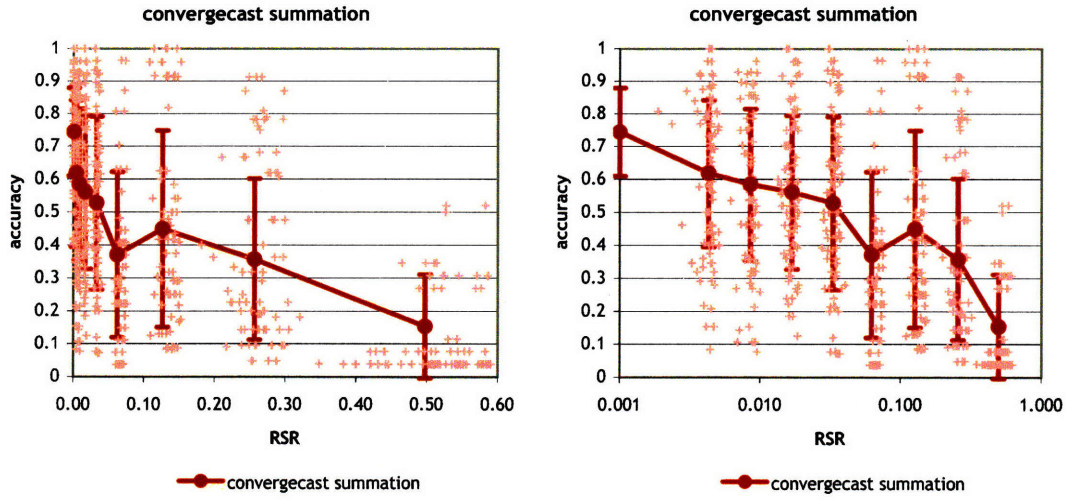


Figure 5-7: Convergecast summation accuracy vs. robot speed ratio. (The same data is shown on both plots. left: linear x-axis, right: log x-axis).

Our accuracy metric does not take into account how many messages from a particular single convergecast were actually accumulated in the summation. As the robots move around, the broadcast and convergecast messages are overlaid and pipelined, and the nature of the algorithm does not preserve individual message routes, as this would require $O(n)$ bits/round. Therefore, the root may be receiving messages from many different convergecasts that have propagated up many different broadcast routing trees at over any one round. In this example, we are computing a summation, so any count, even an erroneous from a previous round count, could increase the accuracy. The only way to separate this error would be to track each individual message through the network and calculate which round a particular convergecast message belongs to, and aggregate it for that round. Our implementation prevents us from doing this, but as long as the errors are uniform, summations over multiple rounds should produce similar results.

5.6 Broadcast Tree Navigation

5.6.1 Problem Definition

The broadcast tree can be used to guide any robot in the configuration to the root using the other robots in the network as navigational aids. We desire an algorithm that uses the broadcast tree for physical routing, to guide any robot from anywhere in the network to the root.

5.6.2 Algorithm

This algorithm requires a broadcast tree. We must also assume that the path between any two neighboring robots is navigable. Since each robot knows its parent in the broadcast tree, any robot can navigate to the root of the tree in a series of steps, by moving to its

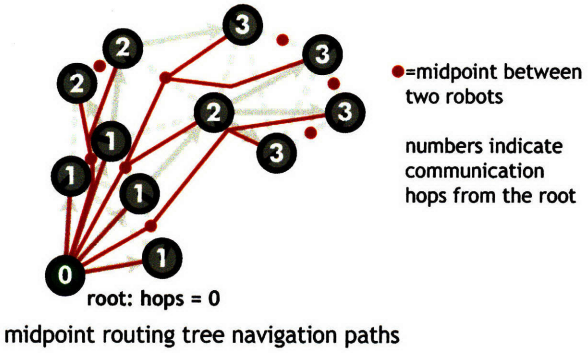


Figure 5-8: An illustration of a midpoint routing tree.

running time (rounds)	n/a
computation per round	$O(m)$
global communications structure	broadcast tree + physical motion
$\mathcal{A}_{\text{BroadcastTreeNavigation}}$	$\frac{s_{\text{optimal}}}{s_{\text{robot}}}$
$\mathcal{C}_{\text{BroadcastTreeNavigation}}$	$O(1)$
$\mathcal{T}_{\text{BroadcastTreeNavigation}}$	$O(diam(G))$

Table 5.4: Properties of the broadcast tree navigation algorithm.

current parent in the tree. In the process of moving towards its current parent, it will move to a new part of the network, and will be able to select a new parent that is closer to the root than its previous one. The cycle repeats until the navigating robot reaches the root of the tree. See reference [73] for an empirical evaluation of this algorithm and reference [63] for a theoretical treatment that proves that progress will always reduce the distance to the root; *i.e.* the broadcast tree does not have local minima.

As in the root vector algorithms, robot a collects all k closer neighbors to the root into the set $\mathbf{P}_a$. It would be sufficient to move towards any neighbor $p \in \mathbf{P}_a$, but this makes no attempt to move along the shortest path or to avoid neighboring robots. Our implementation attempts to reduce the number of collisions by modifying the navigation algorithm slightly to move in between robots on its way to the root. The navigating robot computes the midpoints between all the *cyclically adjacent* neighbors from $\mathbf{P}_a$. Cyclically adjacent neighbors are two robots that are adjacent in the angular ordering of neighbors. The navigating robot selects the closest midpoint and moves towards it. The mobility will cause the neighbors in $\mathbf{P}_a$ to change, and a navigating robot will need to maintain a current set of closer neighbors to maintain progress. If $|\mathbf{P}_a| = 1$, then the robot has only one closer neighbor, its parent, and moves directly towards it. In this case, we rely on the low-level bump-sensing obstacle avoidance behavior to guide the robot around its parent, or any other robot. Once it is clear of the obstruction, navigation towards the root proceeds.

5.6.3 Analysis

The key requirement for success is that the neighbors in $\mathbf{P}_a$ who have fewer hops than the navigating robot are actually closer to the root along the navigable path. Figure 5-3 indicates that this requirement is satisfied in static networks, but Figure 5-5 shows the accuracy of the root vector estimate becomes increasingly uncorrelated with the geometry at high RSRs. We would expect the accuracy of this navigation algorithm to suffer as well.

Figure 5-8 illustrates the path a navigating robot might follow as it moves towards the root. When possible, the navigating robots move towards the closest midpoint between two closer robots on their way to the root. Notice that the midpoint minimizes the chance of collision with any two robots in $\mathbf{P}_a$ by maximizing the separation distance between the navigating robot and both neighbors. However, this benefit comes at the expense of potentially selecting a path that is slightly longer than the optimal path through the configuration. There are locations in the configuration when a navigating robot will only have a single parent. In these situations, the navigating robot moves directly towards its parent.

5.6.4 Physical Accuracy Metric

The physical accuracy of the navigation algorithm is best captured by the *path efficiency*: the ratio of the path actually traveled by the navigating robot to the shortest navigable path possible. For a given start and goal location in the external coordinate frame, the physical accuracy of the broadcast tree navigation algorithm is:

s_{robot} = distance of the actual path traveled by the robot

s_{optimal} = length of shortest navigable path between the start and goal positions

$$\mathcal{A}_{\text{navigation}} \equiv \frac{s_{\text{optimal}}}{s_{\text{robot}}}$$

Paths closer to the optimal are more efficient and "more accurate". Paths that are of infinite length, *i.e.* the navigating robot does not get to the goal position, have an accuracy of 0.

5.6.5 Experimental Results

The accuracy of the broadcast tree navigation algorithm is shown in Figure 5-9, and several example paths from the run where RSR = 0 are shown in Figure 5-10. The algorithm performs well at low and mid RSRs, but then fails completely at a RSR of 0.08 or above, and there were no trials at that speed or greater where the navigating robot was able to reach the root. This is consistent with our intuition that navigation accuracy relies on the broadcast root vector accuracy, *i.e.* the broadcast tree's correlation with the physical structure of the network. When this requirement is not met, navigating robots cannot determine the correct direction to drive in. Qualitatively, the navigating robots move around randomly, moving towards neighboring robots with lower hop counts, but not making effective progress towards the root.

5.7 Summary

The accuracy of all of the algorithms in this chapter decreased as the robot speed ratio increased. None of the algorithms produced practically usable results at the highest speed tested, a RSR of 0.64. The composite graph in Figure 5-11 shows the accuracy of all four

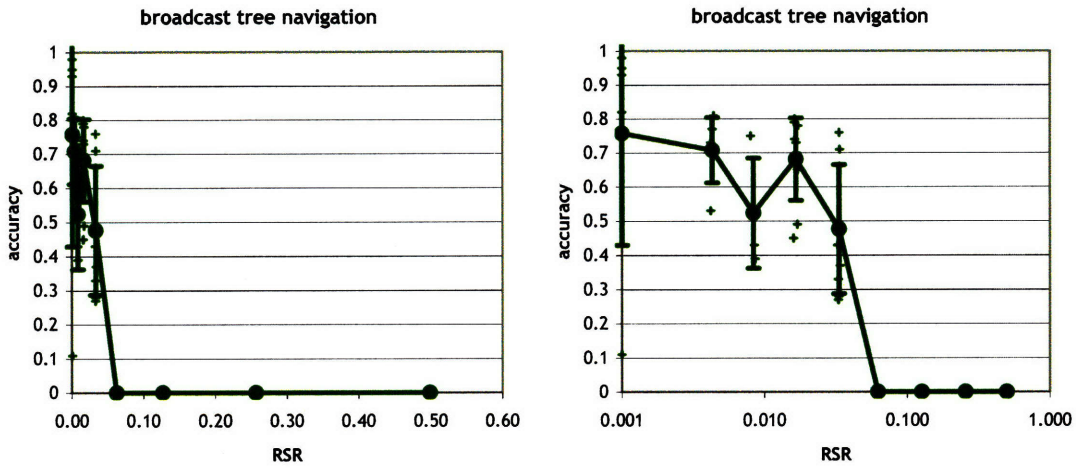


Figure 5-9: Broadcast tree navigation accuracy vs. robot speed ratio. (The same data is shown on both plots. left: linear x-axis, right: log x-axis).

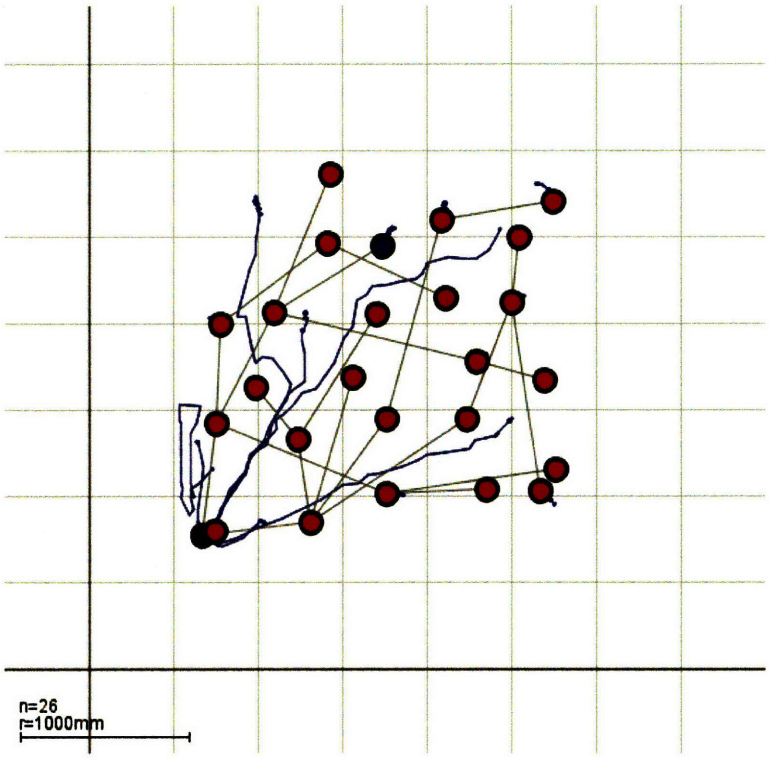


Figure 5-10: Example paths from the broadcast tree navigation algorithm at a RSR of 0. The broadcast tree network is shown with grey edges. Note that it is not the minimal spanning tree, nor is it planar. Using this tree directly for navigation would produce poor results.

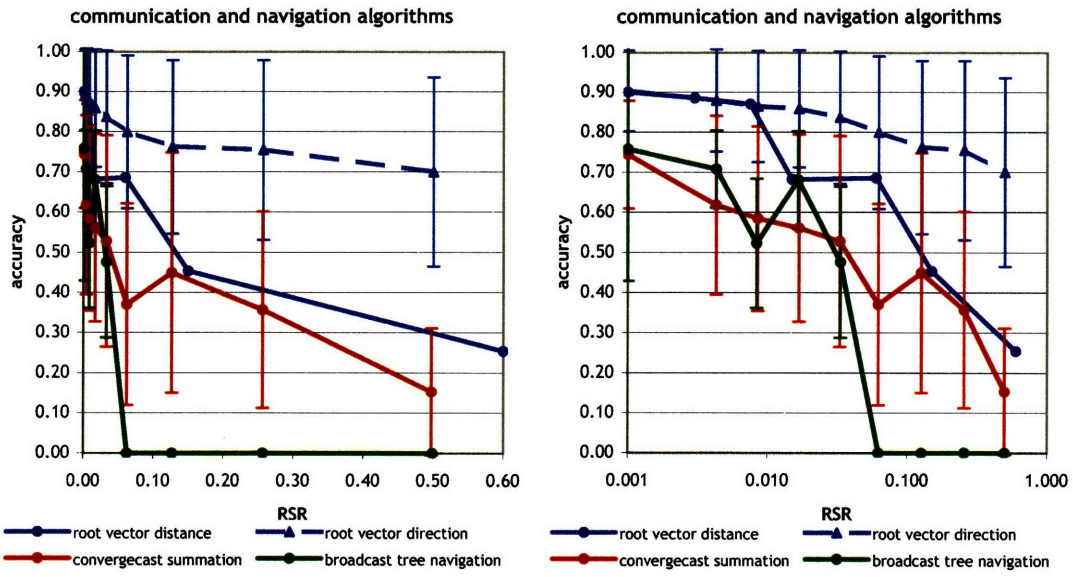


Figure 5-11: Composite accuracy plots for the communication and navigation algorithms from this chapter. The global communications structure of each algorithm is indicated by the color of the line. The blue lines are the accuracy of the root vector magnitude and distance algorithm which both use broadcast tree communication structure. The red line is the accuracy of the convergecast summation algorithm, which uses convergecast global communication structure. The green line is the accuracy of the navigation, which uses the broadcast communication tree communications, but requires the tree to be correlated to the physical geometry with enough accuracy for physical navigation. (The same data is shown on both plots. left: linear x-axis, right: log x-axis).

algorithms from this chapter plotted on the same axes. Roughly speaking, accuracy for algorithms with simpler global communications structure degrades less rapidly than accuracy for those with more complex global communication structure as the robot speed ratio increases. This is not unexpected, as messages have to travel a maximum of 1, $diam(G)$, and $2diam(G)$ hops through the network, for one-hop, broadcast tree, and convergecast global communications structures, respectively. The further the messages have to travel, the more time the network has to change while they are in transit. However, quantitative comparisons between accuracies is limited because each accuracy metric is user-designed and application-dependent. The accuracy metrics used in this chapter were all designed towards quantifying the relationship between the broadcast tree computational data structure and the broadcast tree network geometry, with an emphasis on supporting the navigation algorithm.

Chapter 6

Boundary Detection Algorithms

Joint work with Erik Demaine.

6.1 Introduction

We consider the problem of defining and estimating the boundary of a two-dimensional configuration of robots in a multi-robot system. We desire a boundary that is a "tightly" wrapped contour around the configuration of robots; in other words, one that captures concavities, convexities, and internal voids. We require that the boundary form a connected subgraph of the robot network, so that messages can be routed around it and its properties can be estimated. Finally, we require that the algorithm be fully distributed and require only local network geometry and communications to determine boundary status. Figure 6-1 shows the type of output we desire. Edges in boundary subgraphs are drawn in blue. Robots on the exterior boundary are filled in blue, those on internal voids are filled in green.

A distributed algorithm to estimate the boundary of a multi-robot configuration has many practical applications. We can use a boundary as a formal and practical definition of what is inside and outside the network. Knowing the boundary would allow us to estimate the perimeter of the configuration. For a surveillance application, robots on the boundary can specialize into target monitors, and notify the network when a target has entered or left the tracking area.

In addition to detecting the external boundary, a boundary detection algorithm can find voids in the interior of the configuration. These might simply be voids in the distribution of robots, or might be the footprint of an impassable obstacle in the environment, larger than any one robot could detect individually. These features could be detected, their perimeter estimated, and in the case of voids in the robots' distribution, rectified by moving robots into the empty area.

Boundaries can be used to find *local articulation points*: individual robots whose removal disconnects a local portion of the multi-robot network. These are vulnerable areas in the network, where removing one robot would increase the routing distance between previously nearby robots, or disconnect the network entirely. The third diagram in Figure 6-2 illustrates a network with three local articulation points.

Boundaries can be used for multi-robot configuration control. An internal boundary that is caused by a void in the network can be filled using simple behavior-based motion algorithms. The external boundary can be used to put a "surface tension" on the robots, to

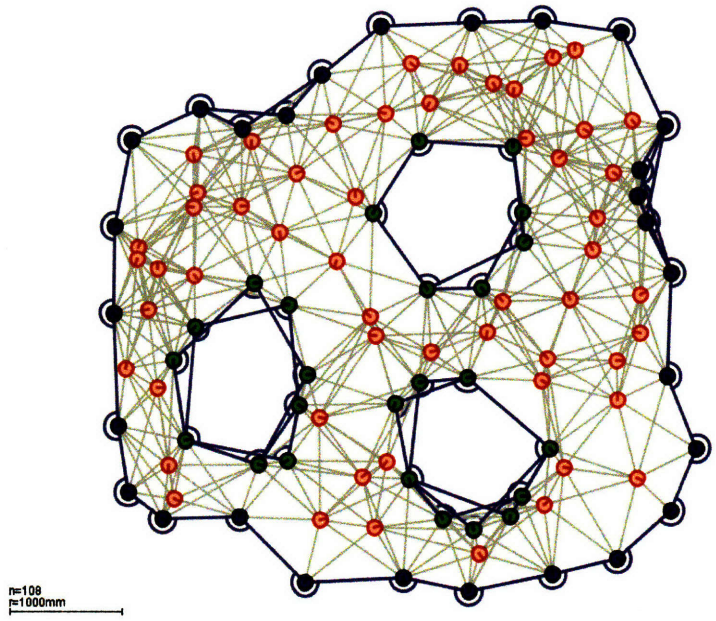


Figure 6-1: We desire distributed algorithms to detect and characterize global boundaries from the local network geometry measured by individual robots. We want to be able to detect concave and convex boundaries, find internal voids, and ensure that each subgraph of the network formed by the boundaries is connected.

help maintain a connected network, force the robots into a more circular configuration, or to compress the configuration into a smaller area. The robots can rectify local articulation points by recruiting nearby robots to add redundancy to the network where needed.

This chapter presents a new boundary detection algorithm called the *cyclic-shape* algorithm. The cyclic-shape algorithm is fully distributed, requiring only the local network geometry available to each robot. We prove certain correctness properties under ideal conditions, and provide a conjecture that cyclic-shape neighbors form a connected subgraph along or near the exterior face of the *configuration polygon*, the polygon formed from the union of all the triangles in the configuration network.

In addition, we present two algorithms to characterize properties of the boundaries in the configuration. The first creates a robust boundary subgraph by passing messages between adjacent boundary robots. The second allows the robots on each boundary to classify the subgraph as an internal void or the external boundary. Figure 6-1 illustrates the output of these characterization algorithms.

6.2 Problem Definition

A natural definition of the boundary of a multi-robot system is the set of robots that will first detect a target approaching from any direction. Assuming that the robot's sensing range and communications range are the same, we can imagine a boundary formed from the union of the communication discs of all the robots, shown in Figure 6-3a. The boundary is defined as any robot who has any part of their disc uncovered, *i.e.*, not overlapping with

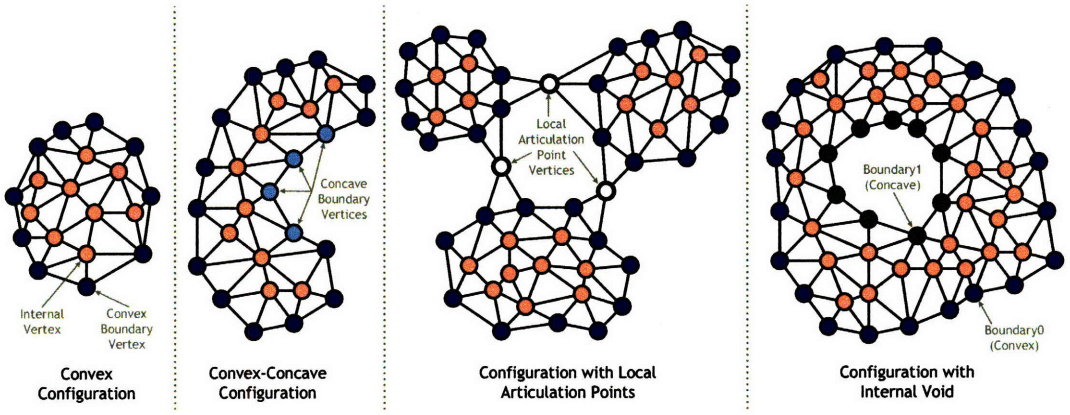
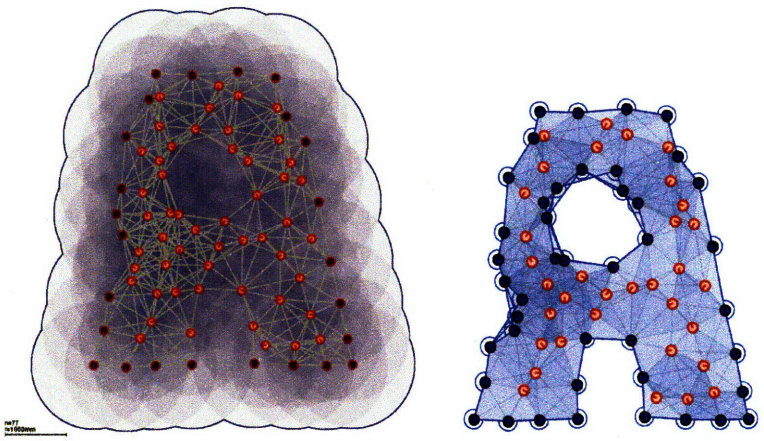


Figure 6-2: This figure illustrates the different type of global boundary configurations we want to distinguish. **Far left:** The simplest boundary configuration is the convex hull. **Middle left:** In general, configurations will also have some areas that are concave. We can define this concavity by using the robot's communication radius as a characteristic dimension to decide what size of a concave feature to admit. **Middle right:.** We want to be able to detect network vulnerabilities called *local articulation points*. These are areas of the graph where disconnecting a single robot can significantly change the network connectivity. **Far right:** We want to be able to detect internal voids, and to be able to distinguish them from the external boundary.



(a) Union of communications radii (b) Union of neighbor triangles

Figure 6-3: Illustrations of two possible boundary definitions. **a:** The boundary can be defined as: "robots whose communication disks contact the outer face." However, when there are concavities in the configuration, this definition does not produce a contiguous boundary subgraph, as can be seen in the lack of boundary nodes in the concavity on the left hand side. Also, this definition does not detect the small void in this example. **b:** The global polygon formed from the union of all the triangles between a robot and each pair of its neighbors can be used to define the boundary. The shading indicates how many triangles are overlapping in a given area. Note the concavity on the left side of the configuration contains robots that are not on the exterior of the polygon. This definition would not include these robots, but they are needed to produce the contiguous boundary that is drawn in blue.

any other disc, and in contact with the outer face of the graph. However, this definition does not always produce a boundary that is a connected subgraph of the network, as a concavity can cause robots to be covered by the discs from their neighbors, as shown on the left-hand side of the configuration. We require a connected subgraph in order to route messages around the boundary for computation and characterization, so this definition is inadequate.

Another way to define the boundary is to construct a polygon from the union of all the triangles formed from each robot to any two of its neighbors, and define the outer face of this polygon to be the boundary. Figure 6-3b shows such a polygon. This definition will include more boundary vertices than the previous approach, and defines the internal void as a boundary. However, it will still not always form a connected boundary subgraph because it can exclude robots that are not on the outer face of the polygon, but are needed for routing. Again, the concavity on the left-hand side of the configuration contains robots that are inside the polygon, and would not be defined as boundaries using this definition, but are needed to form a contiguous subgraph.

This chapter describes a new boundary detection algorithm that is fully distributed, and can detect the concavities and internal voids that these previous two definitions cannot. We pose three problems in distributed boundary detection. First, we want to determine whether a robot is on a boundary using local network geometry and one-hop communications. Second, we want to determine which robots lie on the same boundary, that is, are part of the same boundary subgraph. Third, we want to determine which of these boundary subgraphs are internal or external. We present three distributed algorithms to solve these problems:

Local Boundary Classification: We present the *Cyclic-Shape* algorithm, which lets individual robots determine their local boundary status, and serves as the foundation for the other algorithms. It is more inclusive in its specification of boundary vertices than the definitions illustrated in Figure 6-3, in order to produce a boundary that is a connected subgraph. Unfortunately, this comes at the cost of declaring some vertices to be boundaries that are not, but we will demonstrate empirically that this error is small. We provide a proof sketch of subgraph contiguity under ideal conditions, and empirical results of algorithm performance on a highly mobile group of robots. The algorithm uses $O(1)$ bits/round of communication, runs in $O(m \log m)$ computation per round, and uses $O(1)$ rounds, where m is the number of neighbors. The global communication structure is one-hop communications.

Boundary Subgraph Construction: This algorithm builds a broadcast tree that only propagates through boundary robots. The broadcast tree uses the leader election algorithm described in Section 3.3.2. After the broadcast tree is constructed, all the robots know the ID of the boundary leader, and this ID becomes the ID for the entire boundary. The algorithm uses $O(1)$ bits/round of communication, $O(m)$ computation per round, and $O(diam(G))$ rounds. The global communication structure is broadcast tree communications.

Global Boundary Classification: This algorithm allows each boundary leader to classify its boundary as external or internal. There is only one external boundary, but there can be many internal voids. The algorithm uses $O(1)$ bits/round of communication, $O(m)$

computation per round and $O(diam(G))$ rounds. The global communication structure is broadcast + convergecast + rebroadcast.

6.3 Related Work

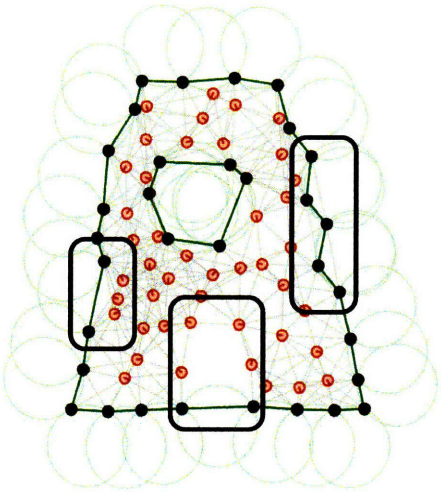
There is much preexisting work on determining boundaries of sets of points. The classic centralized algorithm, the convex hull [96], does not capture concavities in the network. Also, the problem definition operates with the assumption that it is possible to connect any two points with an edge, but our network only supports edges of length $\leq r$. In general, a configuration graph will not have all of the edges required to construct a convex hull. In addition, the convex hull algorithm is centralized, requiring knowledge of the global coordinates of each point to perform the computation.

Fekete et al. present two types of algorithms for boundary detection. Their approaches are fully distributed, using neither global coordinates nor global knowledge, and produce boundaries and other related network structures. One approach takes advantage of the fact that in a dense, uniform distribution of nodes with unit-disk connectivity, nodes on the boundaries will have a smaller degree than those in the interior [33]. This "edge effect" can be used to classify boundary nodes if each node can estimate the global distribution. Their simulations are impressive, with up to 45,000 nodes in the configurations presented. However, this approach requires very high average degree, around 100, and does not use any of the information our model gives us about the local network geometry. In other work [32, 58] they develop a deterministic boundary detection algorithm, again for unit-disk networks with only connectivity information. They construct a clever network subgraph called a "flower" that uses connectivity information a fixed number of network hops from the node under consideration. The disadvantages of this approach are the complexity of these neighborhood-based structures and limitations on the minimum size concavity or region they can detect. And again, the local network geometry model we assume for our robots gives us more geometric information than their model provides, and should allow us to determine boundary status in a more straightforward fashion.

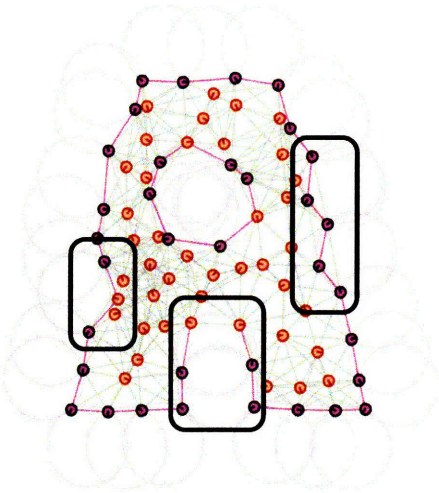
Alpha Shapes

Determining boundaries is a common problem in computer graphics, where it is often desirable to construct a polytope from a collection of points. While all of these algorithms are centralized and use global coordinate systems, they provide insight towards a distributed approach. Since our robot configurations are two-dimensional, we can limit our discussion to algorithms that consider points on a plane.

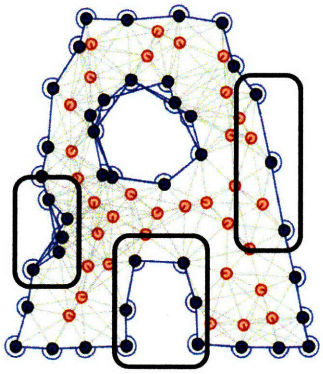
Of these, the α-shape algorithm of Edelsbrunner [29] provides the most inspiration for our approach. Given a set of points on the plane and a characteristic dimension, α, it is possible to construct a well-defined boundary. A point is *α-extreme* if there exists a disc of radius $\frac{1}{\alpha}$ that touches this point and contains no other points. This definition produces different boundary points for different values of α, so it is up to the designer to select the most appropriate dimension for their application. The boundary subgraph can be constructed by taking the union of edges between all pairs of points that a disc of radius $\frac{1}{\alpha}$ can touch and contain no other points. This subgraph is the α-shape, and can be



(a) Global α-shape



(b) Local α-shape



(c) cyclic-shape

Figure 6-4: A comparison of the results of three different boundary detection algorithms. The rounded rectangles highlight areas with differences. **a:** The boundary produced by the centralized α-shape algorithm for $\alpha = \frac{r}{\sqrt{3}}$. Because it has global knowledge of positions and can use any length edge, it adds an edge at the bottom of the figure between two robots whose separation distance is greater than r. Neither of the distributed algorithms can do this. **b:** The boundary produced by a distributed α-shape algorithm that uses local coordinates for $\alpha = \frac{r}{\sqrt{3}}$. It does not form a connected subgraph, as boundary robots are missing in the bottom and left highlighted regions and around the void. **c:** The boundary produced by the cyclic-shape (c-shape) algorithm. The boundary is connected, but there are extra robots in the left region and around the internal void. Comparing the c-shape output to the local and global α-shape algorithms, we note the differences in the concavity shown in the left highlighted region. The c-shape algorithm labeled all these robots as boundaries, the local α-shape algorithm did not label any, creating a gap in the boundary subgraph, and the global α-shape algorithm added a long edge.

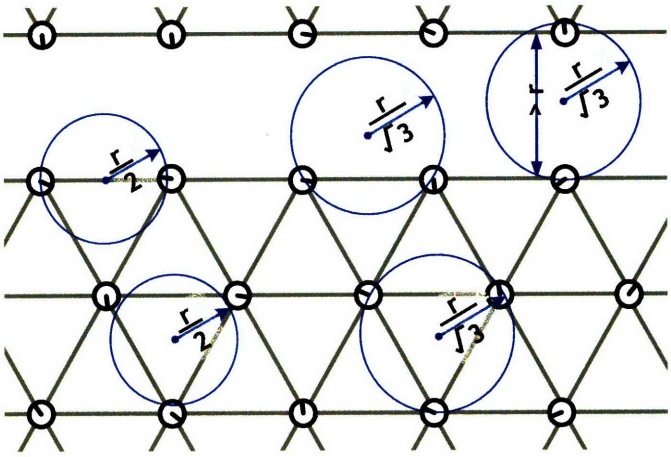


Figure 6-5: In a configuration of robots arranged in a triangular lattice, selecting $\frac{1}{\alpha} = \frac{r}{2}$ will properly detect external boundaries, but will classify all the interior robots as boundaries as well. Selecting $\frac{1}{\alpha} = \frac{r}{\sqrt{3}}$ will correctly classify interior robots and boundary robots. However, the diameter of this circle is larger than the range of the local network geometry ($\frac{2}{\sqrt{3}}r > r$), so a robot's neighbors would not include all of the robots required to make a correct boundary decision. For example, the robots in the upper right should not be boundaries, as a disc of $r = \frac{r}{\sqrt{3}}$ is not empty, but their separation distance is greater than r, so they are not neighbors, and can't determine their boundary status from local information.

interpreted as the boundary of the region containing these points, and has a well-defined interior and exterior. Figure 6-4b shows the α-extreme nodes in green, and the α-shape as green edges. The α-shape is closely related to the Delaunay triangulation, and the edges in the α-shape of a set of points are a subset of the edges from the Delaunay triangulation of the same set of points.

The α-shape algorithm requires the user to select a characteristic dimension, α. In a multi-robot configuration, this dimension can be an arbitrary measure determined by the user to suit the environment, but the communication radius of the robots, r, provides a natural characteristic dimension. Setting $\frac{1}{\alpha} = \frac{r}{2}$ is the obvious selection, but this can produce configurations in which *every* robot is classified as a boundary. In particular, the triangular lattice shown in Figure 6-5 illustrates such a configuration. We can compensate for this configuration and select $\frac{1}{\alpha} = \frac{r}{\sqrt{3}}$. This will correctly classify the interior robots, but extends the range of neighbors each robot needs to consider during execution beyond the range of the local network geometry. So there is no entirely satisfactory value of α.

The third problem is the most serious: the algorithm uses a centralized coordinate system to compute a globally consistent Delaunay triangulation and make globally consistent decisions about which edges to add to the boundary. Because each robot's local network geometry has independent error, individual robots are unable to agree on which edges to use to construct the triangulation. This will lead to neighboring robots making different decisions about their triangulation, and as a result, their boundary status and edges. In the worst case, neighboring boundary robots could both decide they are not boundaries, and disconnect the boundary subgraph. Figure 6-6 illustrates the difficulties of consensus for distributed boundary detection.

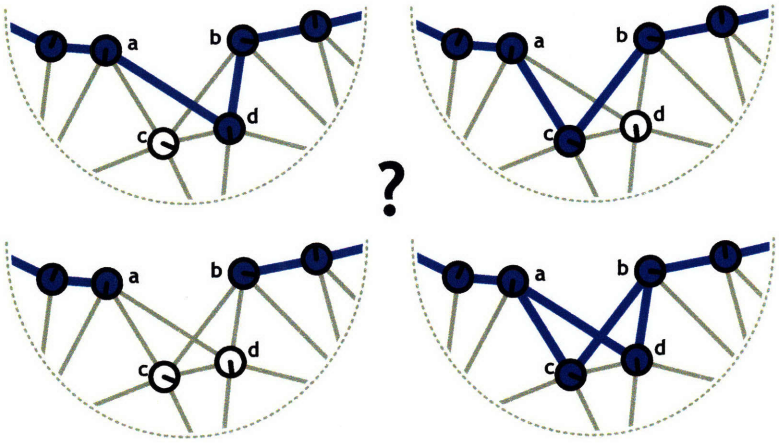


Figure 6-6: In this illustration of a concavity in a boundary, blue edges are part of the boundary subgraph and grey edges are the rest of the network. In order to compute a connected boundary around this network, all four robots $\{a, b, c, d\}$ must make the same decision. However, since the robots measure the positions of their neighbors locally with independent error, the four robots might make different, but locally valid, decisions about which boundary edges to keep. These differences of opinion can produce any of the four global topologies.

We can design a distributed "local α-shape" algorithm that works on the same principles as the global algorithm, but runs independently on each robot and only considers the local network geometry. However, there are many network configurations where our local α-shape algorithm produces a boundary that is not connected, even under the ideal conditions of no sensor error in the local network geometry estimation. Figure 6-4c illustrates a boundary from this algorithm and the kinds of errors it produces.

The fundamental difference between distributed computational geometry in noisy local coordinate systems on robots and centralized computational geometry using global coordinates is the inability to reach a consensuses on geometric relationships between individual robots. Even with a noisy global coordinate system, such as GPS, each robot can measure its own position, and then communicate this information to its neighbors. Even though the measurement might have error, all neighboring robots will have the same measurement error, and come to the same conclusions. In situations where sensor error in the local network geometry can cause two robots to make different decisions, there is no easy way to reconcile these differences, or sometimes even to detect them.

For example, consider the four boundary configurations in Figure 6-6. In order to produce either of the top configurations, all four robots $\{a, b, c, d\}$ must make the same decisions about which edges to classify as boundaries. However, because of local sensing errors, each robot could make a different decision. If we construct a global boundary from the union of each robot's boundary decisions, we can get any of the four configurations. In order to reach a globally consistent graph, this group of four robots will need to reach a consensus, and this means that some robots might have to accept a global boundary structure that is inconsistent with measurements from their local network geometry. A system with perfect local coordinates or any type of global coordinates would only produce one of the top two configurations. Our system can produce any of these cases. Of these, the bottom left one is the most problematic, as it disconnects the network. We would like to

bias our algorithm to produce the bottom right configuration, in order to err on the side of producing a connected subgraph.

Depending on the algorithm and the configuration of robots, local changes to reach consensus can affect the decisions of neighboring robots one hop away from these four, and then by induction, the entire network. This is undesirable for many reasons: it has high communications cost, small local errors that have global effects could result in poor system performance, and distant robots might have to accept geometry that is inconsistent with their local measurements. One of the main challenges of designing distributed algorithms for multi-robot systems is to produce correct, or approximately correct, results that do not require this consensus.

In general, we can classify distributed computational geometry algorithms into four categories based on the kind of localization information they require. An algorithm can require perfect global coordinates, noisy global coordinates, perfect local coordinates, or noisy local coordinates. In the first three categories, robots have consensus of coordinates by construction. Any global system can share positions with neighbors, ensuring that each neighbor has the same estimate, and a perfect local system provides consensus on coordinates by definition. Our model assumes the fourth type, where each robot uses a local coordinate system with noisy position estimates. In systems like this, reaching a global consensus on geometric constructions, such as the boundaries we consider here, or Delaunay triangulations, can be difficult. There is great potential for future work on different types of consensus that might be useful in multi-robot systems. We discuss some of these ideas in Chapter 8.

6.4 Local Boundary Classification

6.4.1 Problem Definition

We want a way to produce a contiguous boundary around the network. We are inspired by the global α-shape, as this construction contains many of the properties we want in our boundary. This section describes the cyclic-shape (c-shape) algorithm. The main intuition behind our approach is that robots who are *not* on the boundary of the configuration are surrounded by a cycle through their neighbors. The second image from the left in Figure 6-7 shows an interior robot and the cycle around it through its neighbors. The c-shape algorithm attempts to construct a particular type of cycle through neighbors of a robot. If it cannot, it labels the robot as a boundary.

In order to avoid inconsistences between a robot's local network geometry and the local geometry of its neighbors (to avoid having to reach a consensus), each robot must perform computation based on its local network geometry only. A robot cannot ask its neighbors for their network geometry or connectivity, as the neighbor's information might be inconsistent with its own local measurements. For example, what should an algorithm do if two neighboring robots say they can communicate, but the robot measures their separation distance to be too large to permit this? Or vice-versa? Or if only one claims to be able to communicate with the other?

To avoid these inconsistencies, we introduce the notion of *inferred edges* to eliminate the need for neighboring robots to share geometry or connectivity information. A robot constructs an inferred edge between any two neighbors when its local network geometry indicates that they are close enough to be able to communicate. This ensures that each

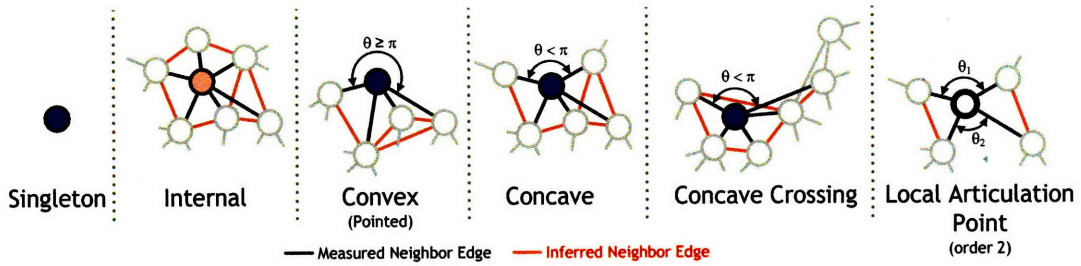


Figure 6-7: Six different local conditions must be handled by the algorithm: Singleton, Internal, Convex, Concave, Concave Crossing, and Local Articulation Point. We define the order of a local articulation point as the number of missing sectors.

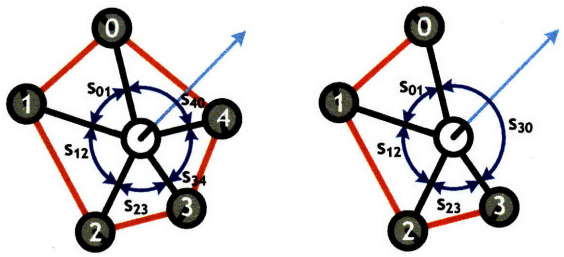


Figure 6-8: We divide the regions around each robot into a series of sectors between adjacent neighbors. We define a *missing sector* as a sector that does not contain an inferred edge, or subtends an angle of π or greater.

robots's local network geometry and its model of its neighbors' one-hop network structure will be consistent, without requiring consensus amongst neighboring robots. This comes at the cost of potentially diverging from the actual network connectivity when the robot makes localization errors, or an obstacle or other error source prevents communication between two neighboring robots.

6.4.2 The Cyclic-Shape Algorithm

We start the discussion of the cyclic-shape algorithm by defining the sectors around a robot. We label a robot's neighbors consecutively, starting from the x-axis of its local coordinate system and proceeding counterclockwise. We divide the region around each robot into a series of *sectors* between adjacent neighbors. Figure 6-8 shows the sectors around a robot as blue arcs. We call sector *missing* if there is no inferred edge between its two adjacent neighbors, or if the angle it subtends is π or greater. Otherwise, the sector is *present*. If a sector is present, we can define a *sector triangle* as the triangle formed between the robot and the two neighbors adjacent to the sector. We define the *configuration polygon* as the union of all the sector triangles from all the robots in the configuration. Figure 6-3b shows a configuration polygon.

The cyclic-shape algorithm looks for missing sectors around a robot, starting at any neighbor and proceeding counterclockwise through all the neighbors. If there are no missing sectors, the algorithm labels the robot as internal. If there is one missing sector, the robot is labeled as a boundary. If there are two or more missing sectors, the robot is labeled as a *local articulation point*, which we will discuss in more detail later.

These local boundary types arise from six classes of local network geometries, shown in Figure 6-7. The main robot in each configuration is drawn with a black outline and black edges, neighboring robots are drawn in grey. Inferred edges are drawn in red, and missing sectors are labeled with black arcs.

A singleton robot is degenerate, and can trivially be classified as a boundary. A robot with no missing sectors is labeled internal. A robot with one missing sector with an angle $\theta \geq \pi$ is labeled as a convex boundary. A robot with one missing sector, but with angle $\theta < \pi$, is labeled concave. A robot missing more than one sector is labeled a local articulation point. We define the order of the local articulation point as the number of missing sectors.

Note that it is possible for a concave robot to have a missing sector, but still be within the configuration polygon. We refer to this as the *concave crossing* case. The illustration in Figure 6-7 shows three concave crossing robots, the main robot with the black outline, and the two neighbors to the right of it. If these robots are not labeled as boundaries, there would be a three-robot gap in the boundary subgraph of this configuration. By labeling concave crossing configurations as boundaries, the c-shape algorithm produces a connected subgraph, but at the cost of labeling more robots than needed as boundaries. The concave crossing case is why the cyclic-shape algorithm considers the neighbors in order, to restrict the algorithm to finding one particular cycle amongst the neighbors. Other cycles might exist around the robot in the local network geometry, but these would classify concave crossing robots as interior.

The cyclic-shape algorithm determines the local boundary classification. It is composed of two functions, FIND-EMPTY-SECTORS and COMPUTE-LOCAL-BOUNDARY-TYPE. The FIND-EMPTY-SECTORS function returns a list of empty sectors around a given robot. The COMPUTE-LOCAL-BOUNDARY-TYPE function takes this list and computes the boundary type.

Algorithm 1 LOCAL-BOUNDARY-CLASSIFICATION(*neighbors*)

```
1: emptySectors ← FIND-EMPTY-SECTORS(neighbors)
2: boundaryType ← COMPUTE-LOCAL-BOUNDARY-TYPE(neighbors, emptySectors)
3: return boundaryType
```

Algorithm 2 FIND-EMPTY-SECTORS(*neighbors*)

```
1: N ← SORT-BY-BEARING(neighbors)
2: emptySectors[] ← ∅                          ▷ Initialize a list of empty sectors
3: for all n_i in N do                         ▷ Search between all neighbors for inferred edges
4:     n_j ← NEXT-COUNTERCLOCKWISE-NBR(N, n_i)
5:     θ ← COMPUTE-COUNTERCLOCKWISE-ANGLE(n_i, n_j)
6:     if θ ≥ π then
7:         ADD(emptySectors, {n_i, n_j, θ})    ▷ Pointed local geometry ⇒ convex
8:     else if (FIND-INFERRED-EDGE-BETWEEN(n_i, n_j) = FALSE) then
9:         ADD(emptySectors, {n_i, n_j, θ})    ▷ No inferred edge, concave boundary
10:    end if
11: end for
12: return emptySectors
```

The FIND-EMPTY-SECTORS algorithm first sorts the neighbors by their bearing in the coordinate frame of the robot running the algorithm. Starting from an arbitrary neighbor,

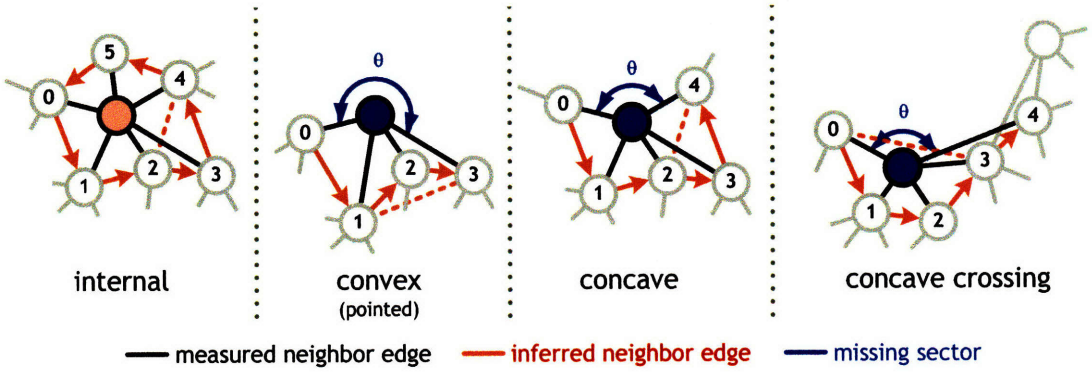


Figure 6-9: Illustration of the cyclic-shape algorithm's progress when considering four boundary types, excluding the singleton case and local articulation points. The algorithm sorts the neighbors of a robot by their local bearing. It then starts at the first neighbor, and check for an inferred edge to the next neighbor. If a cycle through all the neighbors of a robot exists, then that is not a boundary. Note that in the concave crossing case, the robot is inside the interior of the polygon formed by the neighbor triangles, but is still classified a boundary.

Cyclic-Shape boundary detection algorithm properties	
computation per round	$O(m \log m)$
running time (rounds)	$O(1)$
global communications structure	one-hop communications
$\mathcal{C}_{\text{LocalBoundaryClassification}}$	0
$\mathcal{A}_{\text{LocalBoundaryClassification}}$	$\frac{n_{\text{correct}}}{n_{\text{total}}}$

Table 6.1: Properties of the cyclic-shape local boundary classification algorithm

the algorithm looks for an inferred edge to the next cyclicly adjacent neighbor. If there is not an inferred edge, the sector in between the two neighbors is stored in the list of empty sectors. The function terminates when it has checked for inferred edges between all the neighbors.

The COMPUTE-LOCAL-BOUNDARY-TYPE function takes as an input a list of empty sectors and a list of neighbors and returns the local boundary classification. The procedure is straightforward: if there are no empty sectors, the robot is either a singleton or internal. If there is one empty sector, then the robot is either convex or concave (this includes the concave crossing case), and if there are more than one empty sector, then the robot is classified as a local articulation point.

Intuitively, *local articulation points* are robots whose removal disconnects a local portion of the network. This removal does not necessarily disconnect the entire network. We defer a formal definition of a local articulation point to future work.

6.4.3 Analysis

The algorithmic properties of the cyclic-shape local boundary classification algorithm are shown in Table 6.1. The the c-shape algorithm produces a boundary classification in one

Algorithm 3 COMPUTE-LOCAL-BOUNDARY-TYPE(*emptySectors*, *neighbors*)

```
if emptySectors = nil then
    if LENGTH(neighbors) = 0 then
        type ← CONVEX-SINGLETON                    ▷ no empty sectors & no neighbors
    else
        type ← INTERIOR                            ▷ no empty sectors
    end if
else if LENGTH(emptySectors) = 1 then
    if emptySectors[0].θ ≥ π then
        type ← CONVEX                              ▷ one empty sector ≥ π
    else
        type ← CONCAVE                             ▷ one empty sector < π
    end if
else
    type ← LOCAL-ARTICULATION-POINT                ▷ multiple empty sectors
    degree ← LENGTH(emptySectors)
end if
return type
```

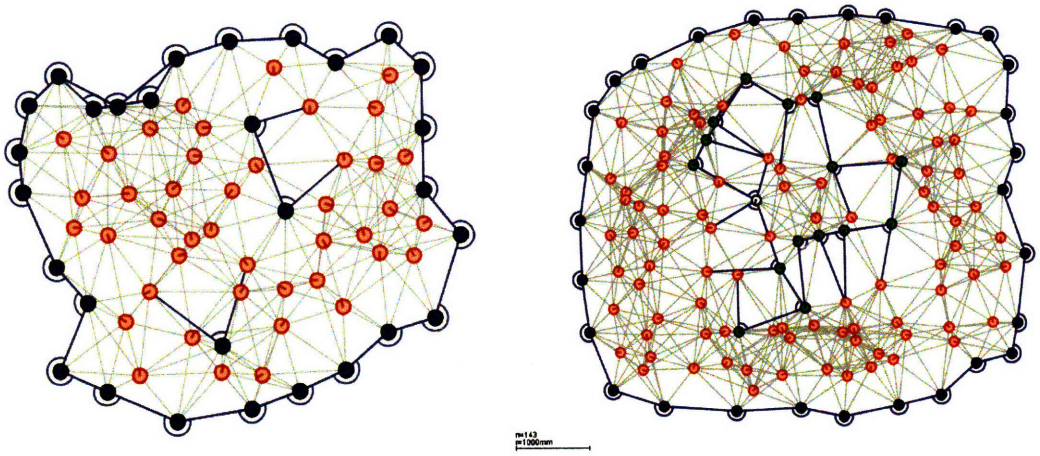
round of computation. During that round, it must sort the neighbors by their bearing, then walk through this sorted list. We can determine whether two neighbors are connected with an inferred edge in $O(1)$ time, so the total computation per round for this local boundary classification algorithm is $O(m \log m)$, where m is the number of neighbors of the robot. The global communications structure is one-hop communications. The communications complexity, $C_{\text{LocalBoundaryClassification}} = O(1)$. Even though the algorithm does not share any information with neighboring robots through public variables or broadcast communications, it still needs to receive announcement messages from neighbors to determine its local network geometry.

When the local network geometry estimates are exact, we can prove that if the c-shape algorithm labels a robot as interior, then that robot is contained within the global configuration polygon.

Lemma 6.4.1 *If a cycle in the local network geometry can be created through inferred edges between neighbors sorted in cyclic order, then the robot is in the interior of the global polygon.*

Proof If there is a cycle through the ordered neighbors in cyclic order, then there is an edge between each adjacent neighbor. Since each neighbor is part of two triangles, there can be no space between adjacent triangles, and the union of these triangles will leave no empty space around the robot. This robot will be within the interior of this local polygon. Since the global polygon is the union of all these triangles from all the robots, the robot will also be in the interior of the global polygon. □

The converse is not always true: the c-shape algorithm can label robots as boundaries when they are within the bounds of the global polygon. We require this property to be able to detect concave crossing boundaries, but it can cause other errors. Figure 6-10 shows two configurations with robots whose local network geometry causes the c-shape algorithm to classify them as boundaries, even though they are within the bounds of the global configuration polygon. Essentially, these errors have a local network geometry that is indistin-



(a) c-shape local classification errors

(b) Cycle of classification errors

Figure 6-10: **a.** Because the c-shape algorithm only uses local network geometry, it is possible for robots that are on the interior of the network to be classified as boundaries. **b.** It is even possible to construct a cycle of erroneously classified boundary robots, making this kind of error more difficult to detect.

guishable from that of a concave crossing configuration. The only way for a robot to distinguish these two cases would be to share information with its neighbors. This information has to be selected carefully, as it could lead us down the path towards requiring consensus between neighboring robots, and by extension, arbitrarily large groups of robots.

One possible approach to reducing these errors is to construct a subgraph between connected boundary robots. Robots that are members of subgraphs that are very small could change their boundary status, but this requires computing the size of the subgraph. Unfortunately, it is possible to construct a connected "chain of errors" of arbitrary length in the interior of a large configuration, even of length greater than the perimeter of the configuration, so this error correction is not reliable. It is also unclear that a boundary subgraph always forms a connected component. We present a conjecture and proof sketch in the next section that states that the external boundary subgraph is always connected, and we have strong belief that the boundary around an internal void is also connected, but it is still possible to form a "cycle of errors" in the interior.

Another approach to reducing these types of errors is to maintain some minimum density of robots, or enforce some additional conditions on local network geometry, such as a Yao graph, or a Gabriel graph, or a NET graph [66, 93, 114]. It might be possible to enforce such a construction at run time with configuration control algorithms. This is an interesting direction for future work.

6.4.4 Physical Accuracy Metric

In order to compute the physical accuracy, we compare the boundary classifications the robots produce to the results of the same algorithm running on a log file of the robots' positions. We accept this external classification as ground truth, and define the physical accuracy of the local boundary classification algorithm as the ratio of correctly identified robots to the total number of robots.

$$\mathcal{A}_{\text{LocalBoundaryClassification}} = \frac{n_{\text{correct}}}{n}$$

.

That this accuracy will not tend to zero in the high RSR case. We assume that robots are able to estimate their local network geometry at the end of each round, and then determine their boundary status with that information. At a high RSR, each robot would carry that estimate with it as it moved, so we would expect that there would be a ratio of boundary robots to interior robots that would be approximately equal to the ratio of the perimeter to the area of the environment:

$$\mathcal{A}_{\text{LocalBoundaryClassification}_{\text{random}}} = \frac{E_{\text{perimeter}}}{E_{\text{area}}},$$

where $E_{\text{perimeter}}$ is the perimeter of the environment and E_{area} is the area. Our experimental environment is a square with sides of length 2.43 m. This produces a perimeter to area ratio of 0.5, and is what we would expect the accuracy to converge to at high RSRs.

6.4.5 Experimental Results

On our actual robot hardware, the cyclic-shape local boundary classification algorithm performs very well on static configurations, with an accuracy of 0.86. Figure 6-11 shows pictures of two static configurations, and simulation sketches to highlight the boundaries. Figure 6-12 plots the accuracy vs the robot speed ratio. The accuracy decays as the RSR increases, and tends to approach the predicted value of 0.5 at high RSRs.

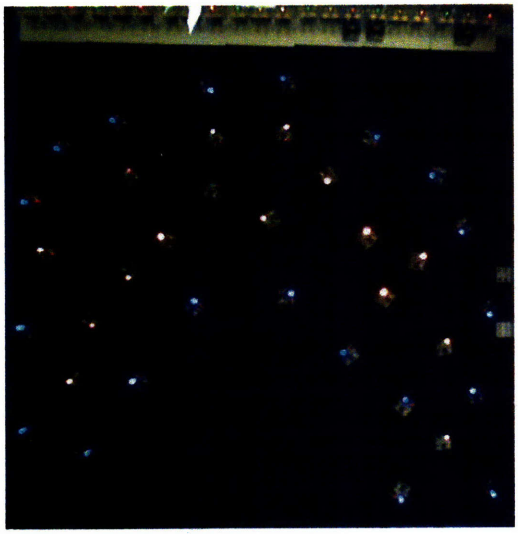
6.5 Boundary Subgraph Construction

Once robots have determined their local boundary classification, we can use this information to estimate global properties about the boundaries in the configuration. In order to do this, we need to construct a subgraph amongst boundary robots to use as a routing network.

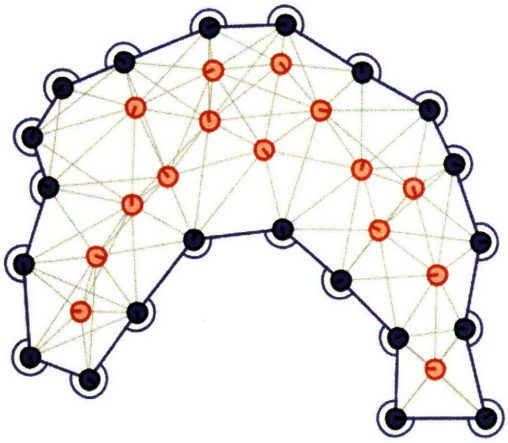
6.5.1 Problem Definition

We want to be able to connect neighboring robots into a contiguous boundary subgraph. Every configuration has at least one boundary, the external boundary. Any other boundaries are caused by internal voids or local classification errors. In this section, we assume any other boundary is caused by an actual void in the configuration, and we neglect erroneously classified boundary robots. We will discuss possible filtering techniques to cope with errors later in the section.

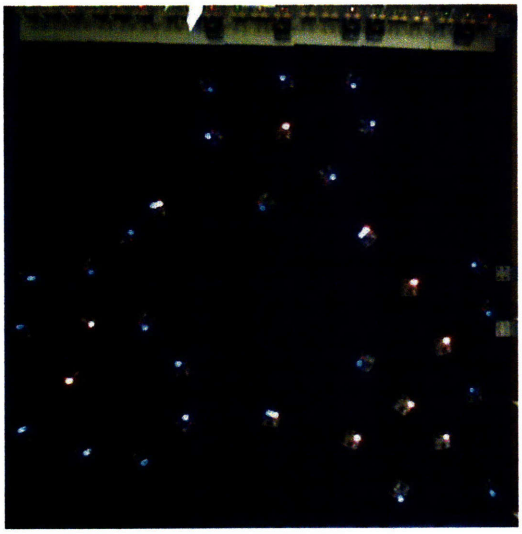
We assert that the the robots labeled as boundaries by the cyclic-shape algorithm form a contiguous boundary subgraph when connected to neighboring boundary robots. We desire an algorithm that can produce a contiguous boundary subgraph for each boundary in the configuration. We want each boundary subgraph to have a single distinguished robot, the root, that can be used to aggregate global information. The boundary subgraph must be self-stabilizing and robust to configuration and population changes.



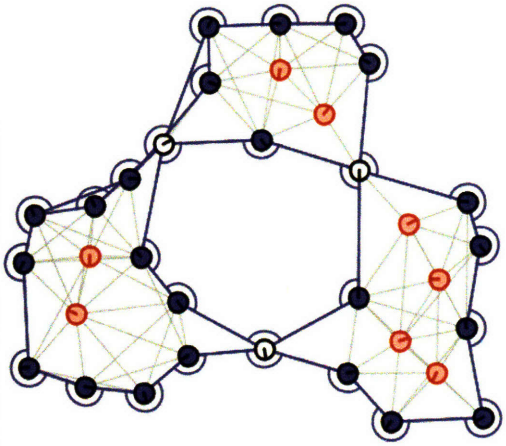
(a) Photo of robots detecting boundaries



(b) Simulation



(c) Photo of robots detecting local articulation points



(d) Simulation

Figure 6-11: The cyclic-shape local boundary classification algorithm running on two static configurations. Internal robots, boundaries, and local articulation points display red, blue and white lights, respectively **a, b.** A configuration with convex and concave boundaries. All of the robots in the image are correctly classified. **c, d.** A configuration with three local articulation points. There are two erroneously classified robots in the picture, on in the upper middle, and one on the lower right.

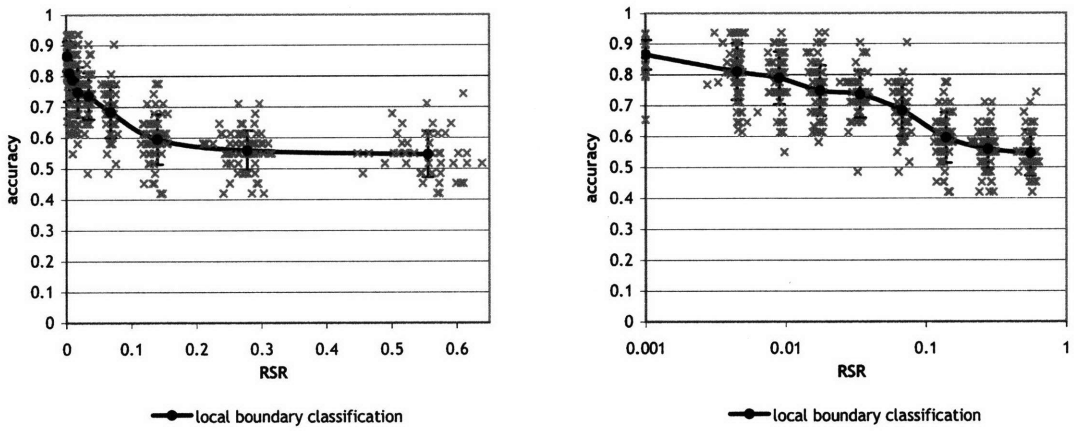


Figure 6-12: Accuracy of local boundary classification algorithm. Average accuracy in static configurations is 0.86. This drops as RSR increases, approaching 0.5, which is the ratio of perimeter to surface area in the 8′ x 8′ workspace. The largest RSR tested occurs when the robots are moving at 80mm/s and communicating every 8 seconds, or 0.64 meters per communication round. This slow update allows them to sense their configuration in one location, but change their state when they are in another. (The same data is shown on both plots. left: linear x-axis, right: log x-axis).

Boundary subgraph construction algorithm properties	
computation per round	$O(m)$
running time (rounds)	$O(diam(G))$
global communications structure	broadcast tree communications
$\mathcal{C}_{\text{BoundarySubGraphConstruction}}$	$O(1)$
$\mathcal{A}_{\text{BoundarySubGraphConstruction}}$	$\frac{n_{\text{correct—subgraph}}}{n_{\text{subgraph}}}$

Table 6.2: Properties of the boundary subgraph construction algorithm

6.5.2 Algorithm

This algorithm requires each robot to have determined its local boundary classification. All boundary robots use the broadcast-tree leader-election algorithm from Section 3.3.2, but with the scope of the messages limited to only boundary robots. [1] A leader will be elected on the boundary in $O(diam(G))$ rounds. The boundary is named after this robot.

6.5.3 Analysis

The subgraph broadcast tree is constructed in $O(diam(G))$ rounds, and the global communication structure is broadcast tree construction. Computation per round consists of evaluating broadcast messages from neighbors, which requires $O(m)$ time per round. There is only one boundary subgraph message sent by each robot, so communication complexity,

[1] Boundary "routing helpers" also participate in the relaying of boundary messages. We introduce this class of robot in the experimental results section.

$\mathcal{C}_{\text{GlobalBoundaryClassification}}$, is $O(1)$ bits/round. Note that the communications used is not a function of the number of boundaries in the network. Because different boundaries are, by definition, non-contiguous, the same broadcast tree message "slot" can be used for all of the boundaries in the configuration, which keeps communications complexity constant for an arbitrary number of boundaries. However, if an algorithm requires that each robot know of the existence of all the boundaries in the network, then $\mathcal{C}_{\text{BoundarySubGraphConstruction}}$ would be output sensitive and grow as $O(b)$, where b is the total number of boundaries in the network.

A boundary subgraph can be used to filter some local boundary classification errors, to let individual robots determine if they should repress their boundary status. Having them repress their status instead of changing status completely can prevent bad interactions, such as neighboring robots oscillating between boundary classifications. When a subgraph is in place, a population-counting convergecast, such as the one used in Section 5.5, can be used to measure the size of the boundary. Boundaries of size three or less are degenerate, and any boundary that is not a cycle is degenerate. A cycle of errors, such as the one shown in Figure 6-10b, could be detected by noting that many neighbors adjacent to empty sectors are not boundary robots, and that many empty sectors are not shared between adjacent robots. In a correctly-formed boundary, each empty empty sector of a robot a will have two adjacent neighbors, b and c, both of which will also be boundary robots. It seems possible to design distributed algorithms to detect and rectify some of these errors. We leave this for future work.

6.5.4 Physical Accuracy Metric

To compute the physical accuracy of subgraph construction, we compare the boundary subgraphs the robots produce to the results of a centralized version of the subgraph algorithm that uses the robot's ground-truth positions. We accept this centralized subgraph as ground truth, and define the *physical accuracy* of the boundary subgraph algorithm as the ratio of correctly identified robots to the total number of robots.

$$\mathcal{A}_{\text{BoundarySubGraphConstruction}} = \frac{n_{\text{correct on subgraph}}}{n_{\text{subgraph}}}.$$

We define a correct robot as one that meets all of the following conditions:

1. Has correctly identified its boundary status.
2. Has a broadcast tree message from the correct subgraph root.
3. Is connected to the root via a path through the network, and each edge along this path has length less than the communications range, r.

The last condition is needed because at high robot speed ratios, a robot's stored neighbor status could differ from the actual physical configuration of neighbors. Robots can acquire a neighbor at the beginning of a round, then move a considerable distance from that neighbor during the round. When the configuration is examined at any point in time, a robot's belief about which neighbors it can communicate with and the actual possible connectivity may be different. When tracing a path back to the root, the accuracy metric fails if two neighbors are currently too far away to communicate.

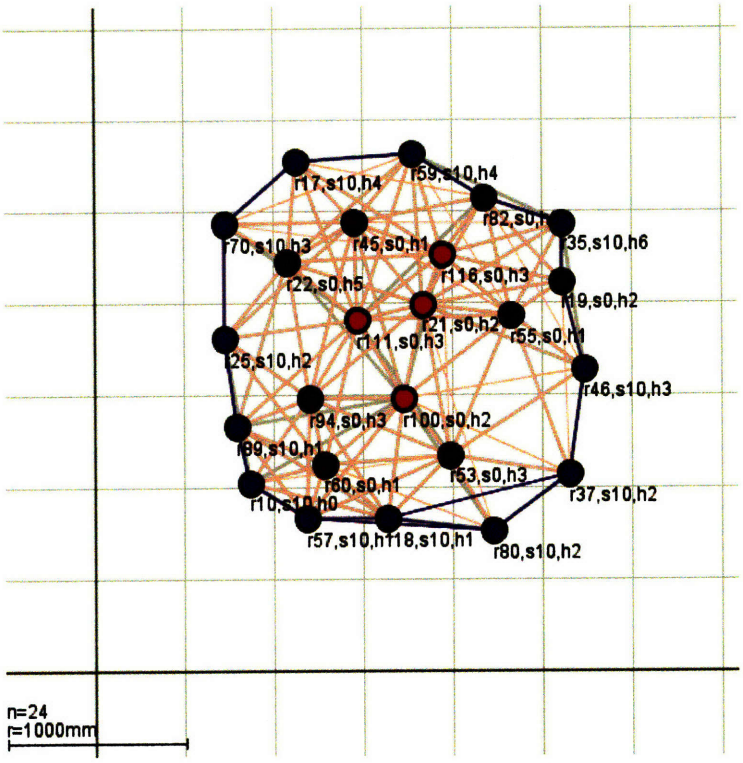


Figure 6-13: Boundary routing helpers add redundancy to the boundary subgraph, which can increase the stability considerably. This illustration is from a log file from an experimental run. Boundary subgraph routing helpers are drawn in purple, regular boundary robots in blue, and interior robots in red. Note that the helpers have patched a break in the boundary near the top of the configuration.

This accuracy metric is very strict, and demands the robots boundaries to almost perfectly match the ones computed by the centralized algorithm. This was necessary to deal with inflated accuracy results caused by the fact the our environment is very small, and can only support one boundary. At high robot speed ratios, any robot that classifies itself as a boundary (which is 50% of them) is likely to receive a boundary subgraph message with the correct root (because there is only one). The only way to distinguish between lucky robots who are in the right place at the right time, and robots who are part of a proper subgraph, is to trace the network back from each robot, through its parents, to try and reach the source. Lucky robots will have broken trees, long edges, and large hop counts. Additionally, if the root robot is incorrectly classified, then accuracy for all robots in that frame will be zero. A larger environment would reduce the odds of a lucky robot receiving a high accuracy.

6.5.5 Experimental Results

The boundary network often has the topology of a line. Removal of any robot or disruption of a single communication link will disconnect the boundary subgraph, causing the algorithm to have poor accuracy on an actual system. To mitigate this, we introduce *routing helpers*: robots who are near a boundary and participate in the routing of boundary

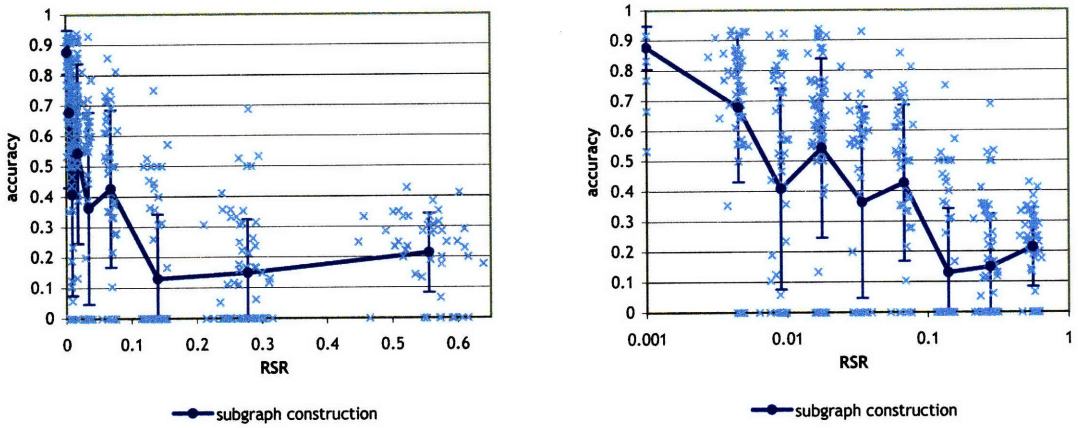


Figure 6-14: Accuracy of boundary subgraph construction algorithm. (The same data is shown on both plots. left: linear x-axis, right: log x-axis).

subgraph messages, but are not classified as boundary robots. Figure 6-13 illustrates an experiment in which a boundary with an error has been patched by a routing helper. We classify a robot as a routing helper if it is within a fixed range d of a boundary robot. For these experiments, we set $d = \frac{r}{2}$.

In evaluating paths along the subtree to the root, a missing robot near the root of the network can break many paths in the subtree, making the accuracy measure highly inaccurate. The data logging system uses a "best effort" approach to deal with poor radio connectivity, and does not receive data from every robot in every frame. This means that robots with poor radio connectivity might not be logged for several frames. At this point, Murphy's law takes over and ensures that the missing robots are always near the root of the tree, which disconnect all their children, and cause wildly inaccurate accuracy measurements. There is no way to correct these kinds of errors without developing a new data collection system. In our analysis, we are forced to accept partial paths to the root, but still evaluate the length of the edges between neighbors, and flag robots with any path edge of length greater than r as incorrect.

Results from these experiments are shown in Figure 6-14. The algorithm accuracy decreases as the robot speed ratio increases. There are many data points with an accuracy of zero. This occurs when the root robot is incorrectly classified, all other robots get an accuracy of zero as well. There is also a slight upward trend at the highest RSRs tested. This is due to the fact that the accuracy metric can be fooled by random configurations. In our small environment, almost all of the boundary robots will eventually receive a subgraph message from the correct boundary root. Checking the lengths of the edges in the paths to make sure they are less than r is not entirely effective, because $r = 1.0$ m in these experiments, but the workspace is only 2.43 m square, so the robots can't get very far from each other, reducing the effectiveness of this accuracy metric.

6.6 Global Boundary Classification

The boundary subgraph can be used by algorithms to compute global properties of the boundaries. For example, the robots can estimate the perimeter of the boundary by using

the root vector distance algorithm from Section 5.3, then using a convergecast to aggregate the distance results on the boundary subgraph root. This section describes an algorithm to classify boundaries as external or internal.

This global classification is important, as the robots may perform different tasks depending on which boundary they are on. For example, robots on an interior boundary might try to eliminate it by moving towards the center of the void. If their motion is blocked by an impassible obstacle, the robots can quantify the object's perimeter. Robots on the exterior boundary can estimate the perimeter, or run configuration control algorithms to help keep the group connected.

6.6.1 Problem Definition

We desire an algorithm to determine whether a given boundary subgraph is the external boundary or an internal void. Any global boundary can be classified into one of four types:

1. **External and convex:** This boundary is on the external boundary of the configuration polygon, and all the robots in this boundary are convex. This boundary is the convex hull of the configuration.

2. **External and mixed:** This boundary is on the external boundary of the configuration polygon, but the local network geometry of its robots is concave and convex.

3. **Internal and concave:** No robots in this boundary are on the external boundary of the configuration polygon, and all the robots in this boundary are concave. This boundary is the "inverse" of the convex hull formed by the void in the configuration.

4. **Internal and mixed:** No robots in this boundary are on the external boundary of the configuration polygon. The local network geometry of the subgraph robots is concave and convex.

To support running configuration control behaviors on the entire boundary, each robot in the boundary subgraph must know the global boundary type. Our algorithm is described below.

6.6.2 Algorithm

The two homogeneous global boundary classifications can be measured by using a convergecast to the subgraph root. Each robot aggregates two quantities, q_{concave} and q_{convex}, which are the number of the two types of local boundary configurations on that robot's subtree. The boundary subgraph root can tally the results to determine whether the boundary is mixed or not. If the boundary is homogeneous, then its exterior/interior status can be determined. A boundary comprised of only convex robots is a convex hull, and can only be external. A boundary comprised of only concave robots can only be an internal void.

But most boundaries are not homogeneous. In this case, determining whether a boundary is internal or external can be accomplished by computing the exterior angle of the polygon the boundary defines. The exterior angle of the outside of a polygon must sum to 2π, while the angle of an internal void will sum to -2π . Each robot uses its local geometry to compute its *turning angle*, and uses a convergecast to aggregate this quantity towards the subgraph root.

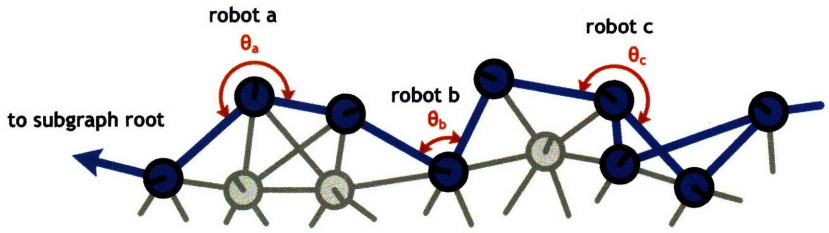


Figure 6-15: The turning angle is $\theta_a - \pi$, where θ_a is the angle swept by the missing sector of robot a.

Figure 6-15 illustrates the geometry of the turning angles and their relation to missing sectors. The turning angle of robot a is $\theta_a - \pi$, where θ_a is the angle swept by the empty sector of robot a. The subgraph root computes the sum of the turning angles, determines the global boundary classification, then rebroadcasts the results back down the subgraph. All three of these operations are pipelined, with the broadcast tree messages, convergecast partial sums, and re-broadcast totals all propagating through the network simultaneously.

The operation of the convergecast for this algorithm is very similar to that of the population summation algorithm used in Section 5.5, but with two key changes. First, the propagation of the broadcast tree is limited to the boundary subgraph, which also limits the convergecast propagation to this subgraph. Second, we compute the sum not over all the children of a robot a, but only one child, selected at random. This is to cope with the additional paths through the boundary subgraph introduced by concave crossing boundaries.

In a concave crossing, there is more than one path through the subgraph. If all the turning angles from all paths are counted, each separate path through a concave crossing would introduces extra concavity into our summation. This error accrues quickly, and can make an external boundary appear to be an internal boundary with around six concave crossings. A solution is for each robot to select only one of its children from the boundary and use this sum in the calculation. Selecting a single child also selects the partial sum of the subtree rooted at that child, so the total sum will be correct.

In order to calculate an exact value of the turning angle through a concave crossing, the robots would need to reach consensus to select a single path through the network, or robots with more than one neighbor with fewer hops would have to provide different partial sums for each potential parent to use. While the second case is implementable, we implement a simpler solution to compute an approximation of the total turning angle through the concave crossing. We sum the angles of the missing sectors instead of the actual angles between the robots used in the paths, which produces an approximate result, but does not require selecting a path or relaying multiple angles to different robots. We will analyze the error more carefully in the next section.

The algorithm requires a broadcast tree on the boundary subgraph to operate. We add three public variables, *parentID*, *turningAngleSum*, and *globalClassification*, to each robot. We examine a particular robot a. Robot a has $k \geq 0$ children in the broadcast tree. We select one of these children at random, say child b. Robot a computes its partial sum in each round by adding its turning angle, $\theta_a - \pi$, to the partial sum of the selected child:

$$a.turningAngleSum = (\theta_a - \pi) + b.turningAngleSum.$$

Global boundary classification algorithm properties	
computation per round	$O(m)$
running time (rounds)	$O(diam(G))$
global communications structure	broadcast + convergecast + rebroadcast
$\mathcal{C}_{\text{GlobalBoundaryClassification}}$	$O(1)$
$\mathcal{A}_{\text{GlobalBoundaryClassification}}$	$\frac{n_{\text{correct—subgraph}}}{n_{\text{subgraph}}}$

Table 6.3: Properties of the global boundary classification algorithm

The partial sum of the turning angle computed on robot a is stored in its public variable, $a.turningAngleSum$, making it accessible to all of its neighbors. After $diam(G)$ rounds, the root robot will have the total sum of the turning angles, and can classify the global structure of the boundary subgraph:

$$c = \begin{cases} \text{external} & \text{if root.}turningAngleSum \geq 0 \\ \text{internal} & \text{otherwise.} \end{cases}$$

The root then re-broadcasts the global boundary classification to all the robots in the subgraph by setting its public variable *globalClassification* to the value computed above. A robot a with parent p in the subgraph broadcast tree relays the global classification as follows:

$$a.globalClassification = \begin{cases} c & \text{if subgraph root} \\ p.globalClassification & \text{otherwise.} \end{cases}$$

This uses the broadcast tree as a routing structure to move global classification information away from the root towards the leaves.

6.6.3 Analysis

Broadcast tree construction, convergecast, and rebroadcast all run concurrently. The subgraph broadcast tree is built in $depth(T)$ rounds, correct turning angle sums reach the root robot after another $depth(T)$ rounds, and the rebroadcast requires another $depth(T)$ to reach the rest of the boundary subgraph. Therefore, the algorithm has a total running time of $3depth(T)$ rounds. This is $O(diam(G))$ rounds, but with a complex global communication structure of broadcast + convergecast + rebroadcast. Computing the sums and relaying them requires $O(m)$ computation per round, and the communication complexity, $\mathcal{C}_{\text{GlobalBoundaryClassification}}$, is $O(1)$.

The algorithm computes an approximation of the turning angle of a single path through a concave crossing using the angles from the empty sectors. Figure 6-16 shows a concave crossing, the angles of the empty sectors in black, and the angles required for an exact sum in red. There are two paths through the concave crossing, *abd* and *acd*, but to compute them correctly would require using the angles drawn in red, which are not empty sectors. We want to compute the expected error of the sum of the turning angles through the concave crossing. We first compute the exact and approximate turning angle sums for each of the two paths.

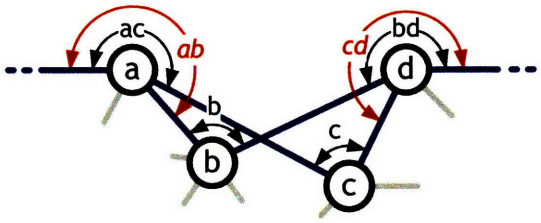


Figure 6-16: The exterior sum angle error for a concave crossing subgraph. If the algorithm sums the results over both paths, it will introduce extra concavity into the total sum. The classification algorithm computes an approximation to the correct turning angle by selecting the path from one child at random. In this figure. compensates by selecting one path at random to sum. This Being agnostic to which path is the "correct" path, because we know that one path must be used for a contiguous boundary subgraph.

The exact turning angle sums are

$$\begin{aligned} E_{abd} &= (ab - \pi) + (b - \pi) + (db - \pi), \\ E_{acd} &= (ac - \pi) + (c - \pi) + (dc - \pi). \end{aligned}$$

The approximate turning angles sums computed from missing sectors are

$$\begin{aligned} A_{abd} &= (ac - \pi) + (c - \pi) + (db - \pi), \\ A_{acd} &= (ac - \pi) + (b - \pi) + (db - \pi). \end{aligned}$$

The next step is to compute the error between these two approaches. Because we don't know which path is the "correct" path, we assume the correct path and approximate path are chosen at random. That produces four combinations of paths:

$$\begin{aligned} e_1 &= A_{abd} - E_{abd} = ac - ab + c - b, \\ e_2 &= A_{abd} - E_{acd} = db - dc, \\ e_3 &= A_{acd} - E_{abd} = ac - ab, \\ e_4 &= A_{acd} - E_{acd} = db - dc + b - c. \end{aligned}$$

It can be shown that $e_1 = e_2$ and $e_3 = e_4$. We estimated the error by sampling the space of convex crossing configurations to produce the distribution shown in Figure 6-17. From this data, we see that each concave crossing subgraph introduces a mean error of 0.272 radians into the calculation of the exterior angle. The worst-case configuration is an exterior composed entirely of concave crossings, one crossing for every three vertices on the perimeter. With an average error of 0.272 radians, it will require 69 robots on the perimeter to accumulate enough error to cause the algorithm to make a wrong decision. For a cir-

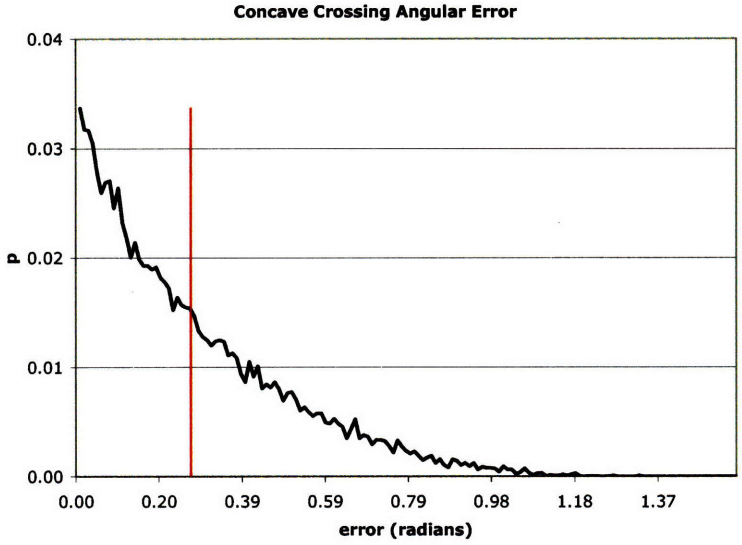


Figure 6-17: We plot the error distribution given by comparing the error computed on either path with the actual concavity on each path, for a total of four cases. The mean error between what the distributed algorithm computes and the actual path is 0.272 radians.

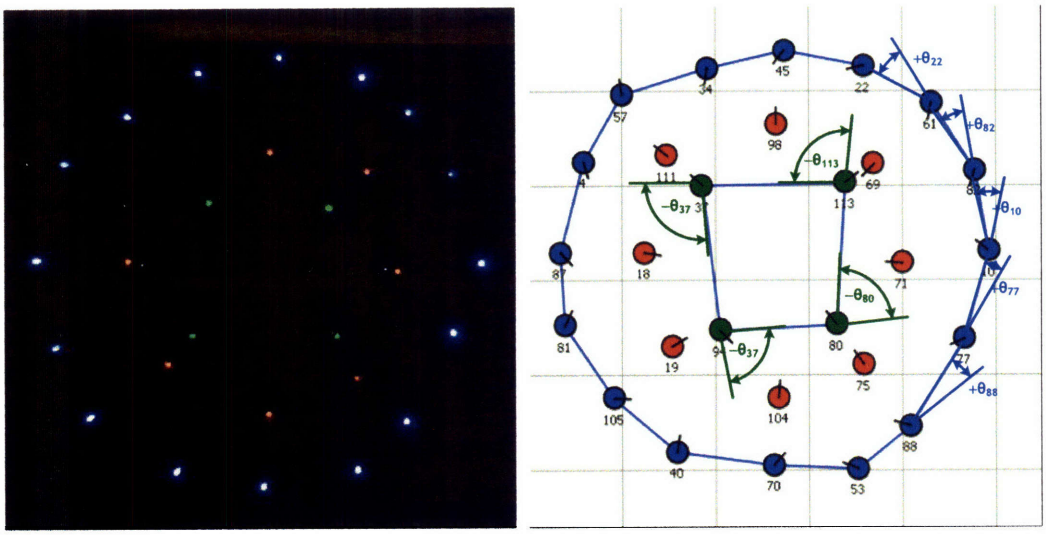
cular configuration, we can expect the number of robots on the circumference to grow as $n_c = 2\sqrt{\pi n}$, so this is a network of 380 robots. In our experiments with around 30 robots, it was uncommon to see more than three concave crossings in any configuration, for a total error of 0.816 radians. This is much less than the $\pm 2\pi$ we expect the global summation to reach, and shouldn't present any classification problems. For very large systems of robots, it might be worth the time to compute the exact sum, but for the systems we tested with, a more serious performance constraint is producing an accurate convergecast.

6.6.4 Physical Accuracy Metric

To compute the physical accuracy of global boundary classification, we compare the recorded global boundary classification of each robot on the subgraph to the results of a centralized algorithm that uses the ground truth robot positions. We define the physical accuracy, $\mathcal{A}_{\text{GlobalBoundaryClassification}}$, of the global boundary classification algorithm as the ratio of correctly identified robots to total number of robots,

$$\mathcal{A}_{\text{GlobalBoundaryClassification}} = \frac{n_{\text{correct on subgraph}}}{n_{\text{subgraph}}},$$

where $n_{\text{correct on subgraph}}$ is the number of correctly classified robots on the subgraph, and n_{subgraph} is the total number of robots on the subgraph.



(a) Image of robots classifying an internal and external boundary.

(b) Screen capture from the same run with turning angle annotations.

Figure 6-18: **a.** Picture of robots running the global boundary classification algorithm. **b.** Annotated screen capture from the same experiment. The robots flashing their blue lights have classified their boundary as external, and the green robots have classified theirs as internal. Robots flashing red lights are not boundaries. The lights are blinking with a sinusoidal pattern. In the picture, brightness variations are evident, and the missing red robot in the upper left has been caught with its lights down. (Which is very embarrassing for a robot)

6.6.5 Experimental Results

Achieving a static configuration with an internal boundary was difficult with our experimental setup. The robots' communication range was too large and the camera's field of view was too small to spread them far enough to minimize communications interference yet still stay within view of the camera. We eventually managed to get two boundaries in this carefully constructed configuration, but it was not very stable, as any connection between the outer and inner boundaries would create a "short circuit", and the boundaries would merge. It was not possible to test internal voids under the same test conditions as the other algorithms, with the BUMPREFLECT behavior in the small environment. I estimate we would need about 2-3 times the workspace area in order to construct a mobile network with an internal void, or we need to modify all 112 robots to have a shorter communications range.

For the BUMPREFLECT experiments, we only evaluated the accuracy of the external boundary classification. Results from the experiments are shown in Figure 6-19. The accuracy of the boundary classification starts very high, but quickly declines as the RSR increases. This is not unexpected, as the convergecast algorithm that underpins the turning angle summation has similar performance. Because having a correct subgraph is a prerequisite for computing a global boundary classification, the accuracy oddities from that metric show up here as well, namely the large number of frames where all the robots had an accuracy of zero, and the slight upward trend towards the end as the configuration became more random.

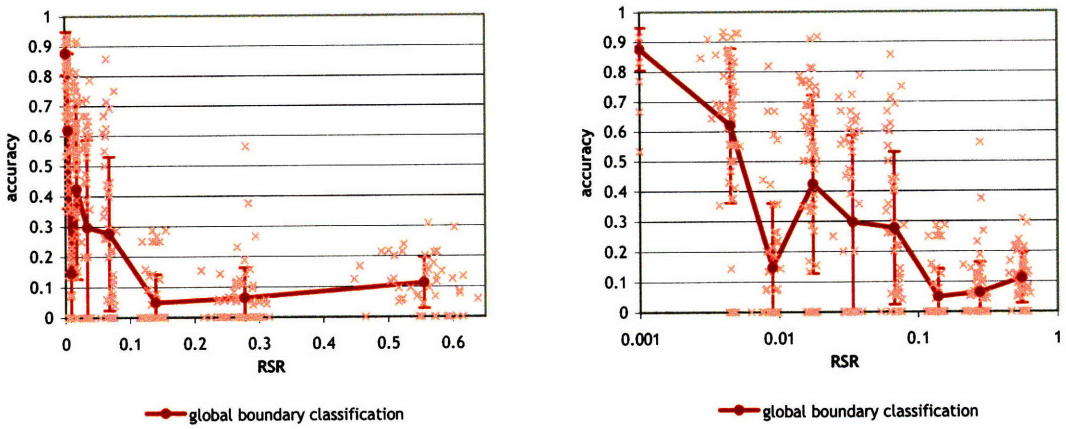
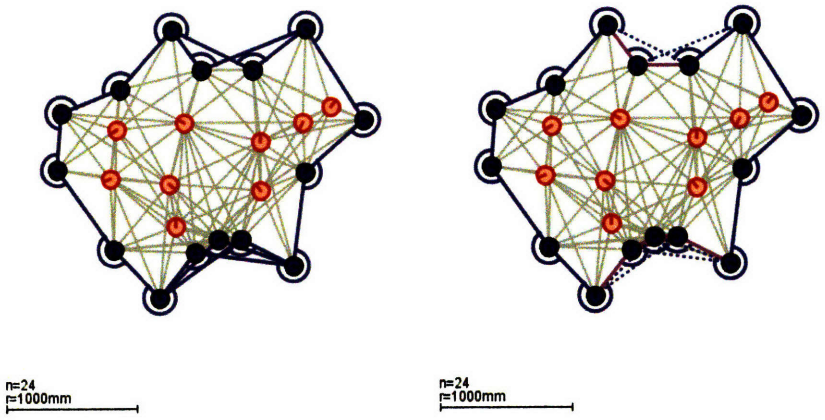


Figure 6-19: Accuracy of global boundary classification algorithm. (The same data is shown on both plots. left: linear x-axis, right: log x-axis).



(a) G Graph with concave crossing edges (b) G' concave crossing edges removed

Figure 6-20: Accuracy of boundary subgraph construction algorithm.

6.7 Proof of Contiguous External Boundary

Earlier, we asserted without proof that the external boundary of a configuration forms a single connected component. This is one of the key features of the cyclic-shape algorithm, so we provide a conjecture and proof sketch that this property is true. There are three key parts to this proof. First, let G_{be} be the external boundary subgraph of graph G. We construct a modified boundary subgraph, G'_{be}, by removing the concave crossing edges and replacing them with edges from G that maintain a contiguous boundary, but do not cross. Second, we show that there exists a path from every node in G'_{be} to a point at infinity without crossing any other edges in G'_{be}. And third, each robot who is on a boundary has at least one empty sector, which is adjacent to two neighbors. We show that any neighbor adjacent to a missing sector is also a boundary robot, and also in G'_{be}. Therefore, by induction, the exterior boundary is a connected component.

The main difficulty with this proof is dealing with the concave crossing case, as these

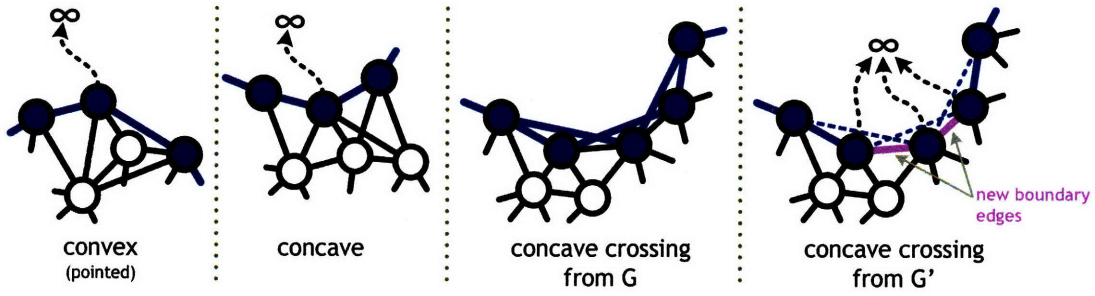
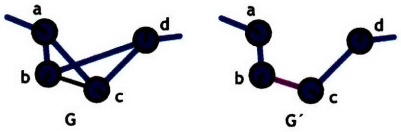


Figure 6-21: We want to show that each node on the external boundary has a path to infinity, but the concave crossing case causes trouble, as it is on the interior of the configuration polygon of the graph G. We construct a new graph, G', with crossing edges removed and new edges defined as boundaries (purple edges). In the graph G', each boundary node has a path to infinity.

robots are surrounded by a cycle of edges, but are still missing a sector. While they are boundary robots, they do not have a path to infinity that is not blocked by an edge in G. Our approach is to define a new boundary subgraph, G'_{be}, with the concave crossings carefully removed. Figure 6-20 illustrates G and G'. We do not consider local articulation points or concave crossings with more than one intersecting edge in the analysis, to simplify the discussion. Note that precluding topologies with local articulation points also eliminates some degenerate topologies with no interior robots, such as the case when the robots are arranged in a line. We defer a complete proof to future work.

Definition Let the *exterior boundary subgraph*, G_{be}, be the boundary subgraph of G that contains a robot $a \in G_{be}$ where a has a path to infinity, *i.e.*, is incident on the outer face of the graph G. There must be at least one such robot $a \in G_{be}$ because there must be at least one robot on the external boundary that has convex local network geometry.

Definition We construct the *non-crossing boundary subgraph*, G'_{be}, by "merging" crossing edges in G'_{be}. Each pair of crossing edges involves four robots, as shown below:



There is only one pair, (a, d), that is not connected in G. We connect the two remaining robots, (b, c), with their edge from G, and remove the original crossing edges, ac and bd, from G'_{be}. This procedure repeats until all crossing edges are removed.

Lemma 6.7.1 *If there is a path between the robots a and d in a concave crossing in G_{be}, then there is a path between them in G'_{be}.*

Proof By construction. The concave crossing in G'_{be} already contains edges ab and cd. By adding the edge bc, we maintain a path between a and d when the edges ac and bd are removed. □

Lemma 6.7.2 *Each robot in G'_{be} has a path to infinity,* i.e., *is on the outer face of the polygon formed by G'_{be}.*

Proof If G is too small or thin to separate the boundaries with interior nodes, then all the robots are boundaries and the lemma is vacuously true. In larger networks, we consider the local network geometry configurations shown in Figure 6-21. Recall that G'_{be} is the exterior boundary, so the convex and concave cases have paths to infinity, as they are on the boundary of the configuration polygon. Once the concave crossing case in G_{be} has been modified into G'_{be}, then these robots have paths to infinity as well. □

Lemma 6.7.3 *Each robot $a \in G'_{be}$ has two neighbors that are also in G'_{be}.*

Proof There are two cases:

1. Robot a is concave or convex. Then a has one empty sector, and two neighbors adjacent to that sector. Since robot a has a path to infinity, and there is an edge between a and its two sector neighbors in G'_{be}, then we can merge these paths and construct a path from a's neighbors to infinity as well.

2. Robot a is part of a concave crossing. This it has two neighbors by construction from the definition of the non-crossing boundary subgraph, and these two neighbors have a path to infinity.

in both cases, there are two neighbors, and both of them are also in G'_{be}□

Claim 6.7.4 *The boundary subgraph, G_{be}, is a single connected component.*

Proof To see that G'_{be} is connected, we pick any robot a, and walk the graph in one direction. Lemma 6.7.1 states that concave crossing merges are connected at the endpoints. By Lemma 6.7.3, each robot has two neighbors that are also on the external boundary because they have a path to infinity by Lemma 6.7.2. Therefore, a tour of the graph starting at any robot must return to that robot, so G'_{be} is connected. □

We have omitted a great deal of computational geometry machinery to sketch this result. Also, we will need to extend the analysis to include local articulation points, and strengthen the connection between G'_{be} and G_{be} and rigorously justify the ability to add edges to G'_{be} in order to connect the subgraph. There may also be a similar result for internal boundaries. It will be somewhat weaker, as it is possible to form an internal cycle of errors; see Figure 6-10. These tasks are left for future work.

6.8 Summary

Figure 6-22 shows the accuracies from the three algorithms from this chapter. When viewed on the same axes, the performance differences of the three algorithms is evident. As with the communication and navigation algorithms, one can only draw limited quantitative conclusions from comparing accuracies across algorithms, as they are subjective and designed to quantify performance for particular applications. However, the accuracy for algorithms with more complex global communication structure degrades faster than the accuracy for those with simpler structure.

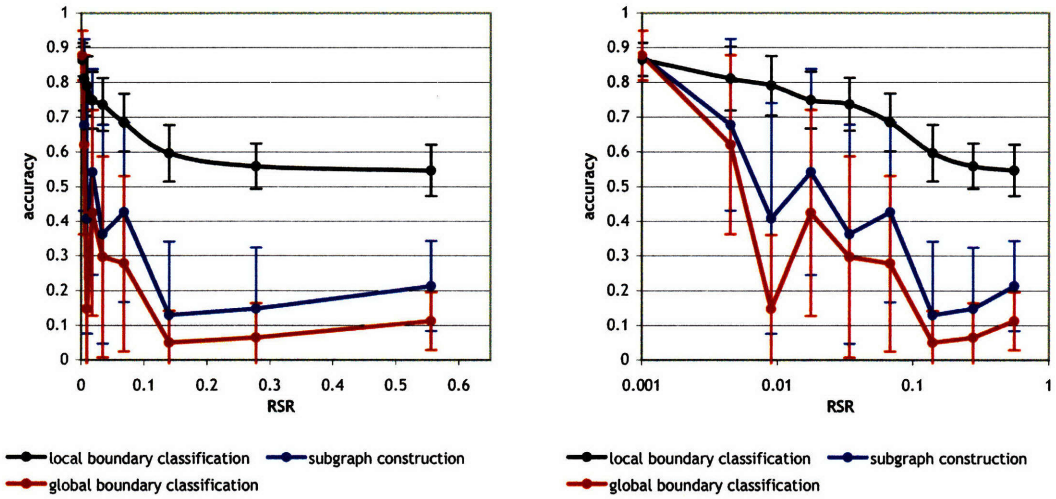


Figure 6-22: Composite accuracy plots for the boundary detection algorithms from this chapter. The global communications structure of each algorithm is indicated by the color of the line. The black line is the accuracy of the cyclic-shape local boundary classification algorithm, which uses one-hop global communication structure. The blue line is the accuracy of the boundary subgraph construction algorithm which uses broadcast tree communication structure. The red line is the accuracy of the global boundary classification algorithm, which uses convergecast global communication structure. (The same data is shown on both plots. left: linear x-axis, right: log x-axis).

These three boundary detection algorithms all accomplish very different tasks. In the next chapter, we discuss three algorithms for dynamic task assignment that all accomplish the same task, but represent different trade-offs in communications complexity, running time, and global communication structures.

Chapter 7

Dynamic Task Assignment

Joint work with Daniel Yamins [72].

7.1 Introduction

Many applications for multi-robot systems will require a large group of robots to perform many different tasks concurrently, with different subgroups doing different tasks. This problem has two components: determining the correct global allocation of robots to perform the different tasks, and then producing the correct assignment of robots to match the desired allocation. This chapter presents distributed algorithms for the second component, dynamic task assignment. We present four distributed algorithms for dividing a swarm of homogenous robots into subgroups to meet a specified global task distribution.

The dynamic assignment of tasks in multi-robot systems has many applications. When ants forage, different workers must simultaneously scout food sources, lay trails, and transport prey back to the nest [41, 45]. In a robotic search-and-rescue mission [8, 10, 71] with 42 robots, a preferred task assignment might be 21 robots to explore the environment, 14 to maintain a communications network, and 7 to mark resource locations. This global distribution of $(\frac{1}{6}, \frac{1}{3}, \frac{1}{2})$ should be maintained over population changes, such as the addition of new robots, or robots leaving the group to recharge their batteries. Even if an entire subgroup is removed, the remaining robots should reassign themselves to preserve the global distribution. The system should also accept new global distribution inputs from a user or a task-allocation algorithm.

The robots are given the desired task assignment in a normalized vector of subgroup percentages. The example above had three subgroups and an input of $\mathbf{P}_{\text{goal}} = (\frac{1}{6}, \frac{1}{3}, \frac{1}{2})$. Our algorithms are limited to groups of homogeneous robots, in which any robot can perform any task. The algorithms are fully distributed, self-stabilizing, and robust to communications errors and population changes. We present four algorithms that span a spectrum of trade-offs between running time, accuracy, communications complexity, and global communications structure.

Algorithm RANDOM-ASSIGNMENT is the simplest solution: robots choose a given task with probability equal to the relative size of that task's subgroup in the target distribution. This algorithm requires no communication and completes immediately. However, it has poor accuracy in small to medium-sized swarms (10-50 robots), and has particular difficulty in forming subgroups that are a small percentage of the total distribution. The other three algorithms are deterministic, and have perfect accuracy under ideal conditions

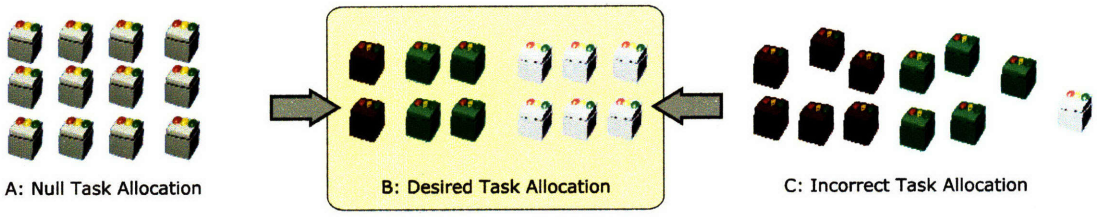


Figure 7-1: Illustration of task assignment for a group of 12 robots. The desired task assignment is given in percentages of red(dark grey), green(medium grey), and blue(light grey), $(\frac{1}{6}, \frac{1}{3}, \frac{1}{2})$. In the left panel, the robots are all unassigned and have no task. In the right panel, the task assignment is incorrect. A task assignment algorithm will drive the system to the distribution illustrated in the center panel with 2 reds, 4 greens, and 6 blues. Robots with similar tasks are placed near each other for illustrative purposes - this is neither a result of nor a requirement for algorithm execution.

of no message loss and stable network topology. The SEQUENTIAL-ASSIGNMENT algorithm uses a single broadcast communications flood to assign tasks to one robot at a time, but has a long execution time. The SIMULTANEOUS-ASSIGNMENT algorithm uses a broadcast flood for each robot, achieving a short execution time by using much more communications bandwidth. Finally, the TREE-BASED-ASSIGNMENT algorithm is a compromise between SEQUENTIAL-ASSIGNMENT and SIMULTANEOUS-ASSIGNMENT balancing communications use and running time. It uses a single broadcast flood to build a spanning tree, then collects the current task distribution onto a single robot. This robot computes the "retasking" required to reach the goal distribution, then broadcasts these changes to the group. All of the algorithms are self-stabilizing, making them robust to communications errors and population changes. When implemented on our test system, the algorithms generally performed to predicted levels of communications usage, running time, and assignment accuracy.

7.2 Related Work

The first stage of the task allocation problem, determining the correct global allocation of robots to perform the different tasks, has been studied extensively by Gerkey and others [37, 38]. The problem we study here, that of achieving a target distribution, has been addressed using probabilistic threshold models [13, 57, 82]. But these techniques do not provide precise and variable control over global task distribution, and provide no guarantee for success in small groups. Also, when the number of tasks exceeds a small number (often 2 or 3), probabilistic threshold methods often have multiple steady states which can trap systems away from a desired task distribution [13]. What is needed is an algorithm which deterministically drives the task distribution to the desired global proportions, independent of other system dynamics.

7.3 Problem Description

We desire a distributed algorithm to solve the task assignment problem. Let k be the total number of tasks. Given a *goal distribution*, a k-dimensional vector $\mathbf{P}_{\text{goal}} \in \mathbb{R}^k$ whose elements sum to 1, we wish to assign each of the n robots one of the k tasks so that the assignment of tasks amongst robots best approximates $\mathbf{P}_{\text{goal}}$.

At any given time, the system possesses a task assignment $\mathbf{A} = (a_1, \ldots, a_k)$ in which a_i is the number of robots assigned to task i, and $n = \sum_{i=1}^{k} a_i$. We also define a task distribution $\mathbf{P}$, which is a normalized version of $\mathbf{A}$: $\mathbf{P} = (a_1/n, \ldots, a_k/n)$, where the elements of $\mathbf{P}$ sum to 1. The goal of a task assignment algorithm is to drive the distribution $\mathbf{P}$ to converge as close as possible to the goal distribution, $\mathbf{P}_{\text{goal}}$. The algorithms must be self-stabilizing, and able to recover from arbitrary changes in the total number of robots n, message loss, or changes in the topology of graph G, as long as the network remains connected.

7.4 Shared Algorithms

All four task assignment algorithms must solve the same basic problem: selecting the best integer representation $\mathbf{A} = (a_1, \ldots, a_k)$ of the real-valued goal task distribution $\mathbf{P}_{\text{goal}}$ applied to n robots. The closest integer number of robots to assign to group i is $|a_i| = round(np_i)$. While this calculation tells us how many robots we need in each group, it doesn't tell us which robots to put into which groups. We need an algorithm that allows each robot to select its own task, and be certain that other robots aren't selecting their tasks in a way that would produce the incorrect global assignment. Solutions that require more global information on each robot are less attractive, because of the communication resources required to estimate and distribute this type of information.

7.4.1 Global Task Bining

Our general approach is to sort the n robots by their robot ID, then use each robot's place in the sorted order to select its task. This gives each robot a unique mapping into the goal distribution, and ensures that the global assignment will be correct. We call this process *task bining*.

Recall that the goal distribution is $\mathbf{P}_{\text{goal}} = (p_1, \ldots, p_k)$, and the current integral assignment is $\mathbf{A} = (a_1, \ldots, a_k)$. The bining algorithm lets an individual robot select its task without computing the entire goal assignment. Let x be the position of robot b in the sorted order of all robots. If $x/n \in [p_1 + \ldots + p_{i-1},\ p_1 + \ldots + p_i]$ then robot b selects task i. The global goal assignment, $\mathbf{A}_{\text{goal}}$, can be determined by bining each robot, determining its task t, then incrementing $a_{(goal)t}$.

The running time for a single robot to select its task is $O(k + n \log n)$. The robot must compute x, then search the k task intervals in $\mathbf{P}_{\text{goal}}$. The running time to compute $\mathbf{A}_{\text{goal}}$ is $O(nk + n \log n)$. This performance is acceptable, as k will probably be a small constant and we assume each robot can operate on $O(n)$-sized data structures. However, the algorithm requires each robot to know its position in the ordered list of all robots and the total number of robots. These global quantities can take be expensive to estimate, in terms of rounds of computation and communication bandwidth.

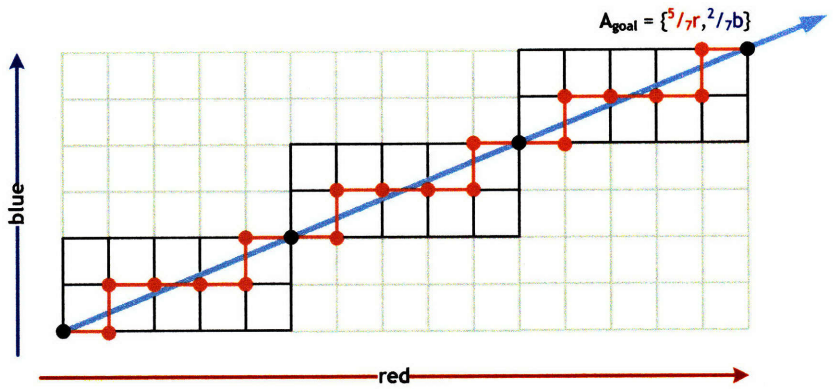


Figure 7-2: Illustration of sequential bining. The desired task assignment is given in the vector $\mathbf{P}_{\text{goal}}$. In this example, $\mathbf{P}_{\text{goal}} = (\frac{5}{7}, \frac{2}{7})$, which represents $\frac{5}{7}$ robots performing the red task and $\frac{2}{7}$ robots performing the blue task. The path repeats every 7 steps along the path, at the points where the real-valued vector $\mathbf{P}_{\text{goal}}$ intersects the integral grid. These repeating sections are the *minimal representation* of the integer path. The grid cells around each minimal representation is highlighted in black.

7.4.2 Sequential Task Bining

It is possible for a robot a to select a task without knowing the total number of robots, if it knows its order, x, on the sorted list of robots. To see this, we visualize the integral representation of a goal task assignment $\mathbf{A}_{\text{goal}}$ as a path starting at the origin, then traveling through integral space of dimension k such that each step in the path is the closest point to the goal distribution vector, $\mathbf{P}_{\text{goal}}$.

Figure 7-2 illustrates this idea. If robot a knows its sorted order, x, it can determine its place on this path and compute the global goal assignment, $\mathbf{A}_{\text{goal}}(x)$, for the case where $n = x$. Robot a can select its task by comparing $\mathbf{A}_{\text{goal}}(x)$ to $\mathbf{A}_{\text{goal}}(x-1)$, which it the global assignment for the previous place on the path. The difference between the two global assignments is robot a's task.

Because the robots are sorted by their ID, robot a's place on the path, and its task, will not change as robots with higher IDs select their tasks. Robot a can select its task as soon as it knows its order. The SEQUENTIAL-ASSIGNMENT algorithm takes advantage of this fact in its execution.

We can simplify further by using the fact that any distribution $\mathbf{P}_{\text{goal}} \in \mathbb{Q}^k$ possesses a *minimal representation*, the sequence of integers $\mathbf{v} = (v_1, \ldots, v_k)$ of minimal sum $V = \sum_i v_i$ such that $\mathbf{P}_{\text{goal}} = \mathbf{v}/V$. This minimal representation represents the first point in integer coordinates that lies along the vector $\mathbf{P}_{\text{goal}}$. These minimal representations are shown in black grid line in Figure 7-2, with black dots to separate them. A robot can compute its position in the minimal representation by knowing the sum of the minimal representation and its position in the sorted order. From this, it can select its task.

Figure 7-3 illustrates a minimal representation of $\mathbf{P}_{\text{goal}}$ in blue and the closest integral path in red. Starting at the point on the lower left, each step along the path is the closest integral assignment of robots to $\mathbf{P}_{\text{goal}}$. The integral path need only be computed for the minimal representation, as it repeats every V steps. Let x be the position of robot a in the

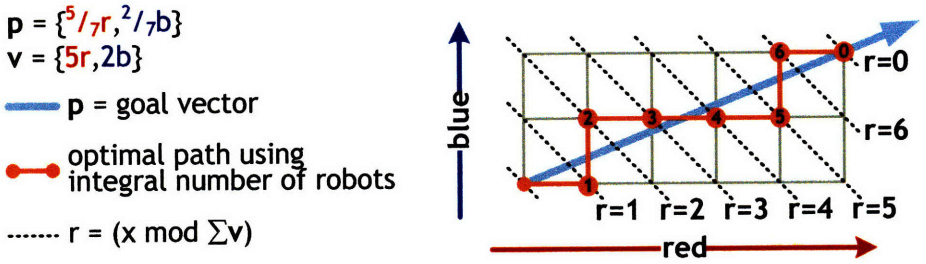


Figure 7-3: Illustration of the *minimal representation* from the sequential bining from Figure 7-2. The desired task assignment is again $\mathbf{P}_{\text{goal}} = (\frac{5}{7}, \frac{2}{7})$.

sorted order, and V be the sum of the minimal representation. A robot can compute $r = x \mod V$, which its position on the integral path. From this position, a robot can compute the best integral assignment for itself and its predecessor at position $r - 1$ on the path. The difference between these two assignments is the task assignment for robot a.

7.4.3 Diameter Estimation

The SEQUENTIAL-ASSIGNMENT and TREE-BASED-ASSIGNMENT algorithms require each robot to obtain an estimate of the diameter of the network to determine how long to wait for a broadcast tree message to reach all robots. In the SEQUENTIAL-ASSIGNMENT algorithm described in 7.5.3, this is to ensure that robots wait long enough to elect a leader before moving to the next stage. The TREE-BASED-ASSIGNMENT algorithm described in section 7.5.5 uses a similar delay to synchronize robot state transitions to within a diameter period across the network. Under ideal conditions, messages travel one hop per round, so the minimum amount of time broadcast tree construction can take is $diam(G)$ rounds.

The diameter is estimated by starting a broadcast tree from any node $a \in G$, and noting that $diam(G) \leq 2D$ where D is the depth of the tree rooted at a. This will give us an overestimate of the graph diameter if the root node a is close to one side of the network, but this overestimate is sufficient for our algorithms, as it will only increase the waiting time. Since we can select node a arbitrarily, we usually select the robot with the lowest ID in the network using a leader election algorithm.

We add a public shared variable[1], *depth*, to each robot. If a robot is a leaf, it sets *depth* to be the number of hops it is from the root. All other robots set their *depth* variable to the maximum of the *depth* variables of their children. For example, the parent of a leaf robot with *depth* = 4 would set its *depth* variable to 4. Its parent would do the same. in this way, the maximum depth measurement propagates towards the root, using the spanning tree as a routing structure. Whenever two subtrees meet, the larger depth is relayed towards the root. The root can then rebroadcast the global maximum depth to the rest of the robots. This process continues indefinitely, to allow the depth measurement to adjust to changing network topology.

Care must be taken during initialization, as the broadcast tree is not constructed yet. It takes $diam(G)$ rounds to build a broadcast tree. In order to prevent underestimating

[1]See Section 3.3.1 for a description of our inter-robot communications system.

the depth of the tree, a robot that has not received a broadcast tree message will report a depth of $n_{\max}$ (the global constant for the maximum allowable number of robots) as its (over-)estimate for the tree diameter.

In the pseudocode in this chapter, input arguments in <brackets> are public shared variables. They are global in scope, so writes to them from within the algorithm will be visible to all other functions and to all neighbors.

Algorithm 4 DIAMETER-ESTIMATION(*broadcastMessage*)<*depth*>

```
1: if ISLEAF(broadcastMessage) then
2:     depth ← HOPS(broadcastMessage)
3: else
4:     depth ← MAX(CHILDREN(broadcastMessage).depth)
5: end if
6: if ISROOT(broadcastMessage) then
7:     diamEstimate ← 2depth
8:     SOURCE(broadcastMessage)
9:     ATTACH(broadcastMessage, diamEstimate)
10: else
11:    diamEstimate ← GET(broadcastMessage, depth)
12: end if
```

In the pseudocode in algorithm 4, the *broadcastMessage* is a global flood that creates a spanning tree. This tree is used to find parent and children relationships amongst the neighboring robots. Robots that are leaves on the spanning tree initialize their *depth* variable to be the number of hops the broadcast message has traveled to reach them. All other robots compute the max of the depth estimates from all of their children in the tree. This effectively propagates the depth estimate towards the root. The root robot computes the diameter estimate, then rebroadcasts this estimate back down the tree by attaching it to the broadcast message. This provides all the robots with the diameter estimate.

In a static configuration with no message loss, the broadcast tree takes $diam(G)$ rounds to construct, and the maximum distance in the network any depth estimate has to travel to reach the root is also $diam(G)$ hops. This gives the root a correct estimate in $2diam(G)$ rounds, which can then be re-broadcast with all other robots in another $diam(G)$ rounds, for a total computation time of $3diam(G)$ rounds.

7.5 Dynamic Task Assignment Algorithms

This section describes the four dynamic task assignment algorithms. We add the public variable *task* to each robot. The *task* variable is an integer and can assume a value between 0 and k. If the robot has selected a task, the value will range from 1 to k. A value of 0 indicates the robot is currently unassigned.

7.5.1 Accuracy Metric

We define the error between the current and desired assignments to be the L2 norm, $e = \sqrt{\sum_{i=1}^{k}(\mathbf{P_i} - \mathbf{P}_{(\text{goal})i})^2}$. The accuracy of each of the algorithms is given by $a = 1 - e$.

7.5.2 Random Assignment

Algorithm

In the RANDOM-ASSIGNMENT algorithm each robot draws a random number $x \in [0,1]$ uniformly and bins it with respect to $\mathbf{P}_{\text{goal}}$. This uses the global bining algorithm described above, but instead of using the sorted order as an input, the random value x is used instead. As above, if $x \in [p_1 + \ldots + p_{i-1},\ p_1 + \ldots + p_i]$ then the robot selects task assignment i. The running time for this algorithm of $O(k)$, depending only on the time it takes to choose and bin x. No inter-robot communication is required, so the global communication structure is no communications.

Analysis

Let X_i be the random variable defined by the relative percentage of robots in task i after all n robots have chosen as above, so that $X = (X_1, \ldots, X_k)$ is a random k-dimensional vector whose entries sum to 1. The variable $n \cdot X$ is distributed according to the $(k-1)$-dimensional multinomial distribution with sample size n and mean $\mathbf{E}[X] = n \cdot \mathbf{P}_{\text{goal}}$. Hence, X itself is distributed according to a $(k-1)$-dimensional multinomial distribution in which the domain has been normalized by n, the number of robots. X has expected value $\mathbf{E}[X] = \mathbf{P}_{\text{goal}}$ and covariance matrix

$$\Sigma_{\mathbf{P}_{\text{goal}}} = \mathbf{E}[(X - \mathbf{E}[X])(X - \mathbf{E}[X])^T] = \frac{1}{n}(\mathbf{D}_{\mathbf{P}_{\text{goal}}} - \mathbf{P}_{\text{goal}} \cdot \mathbf{P}_{\text{goal}}{}^T),$$

where $\mathbf{P}_{\text{goal}}{}^T$ is the transpose of $\mathbf{P}_{\text{goal}}$, a k-dimensional row-vector, and $\mathbf{D}_{\mathbf{P}_{\text{goal}}}$ is the diagonal matrix whose i-th diagonal element is p_i. Hence, accuracy improves quickly as n increases. However, in a group of 40 robots, a task requiring 5% of the total robots stands a nearly 13% chance of receiving no assignment. If this task is required for the global application, the entire mission will fail. Many practical systems are too small for such probabilistic methods to be reliable.

Algorithm 5 DTA-RANDOM-ASSIGNMENT($\mathbf{P}_{\text{goal}}$)

```
1: x ← RANDOM[0 ... 1]
2: s ← 0
3: for all p_i in P_goal do
4:     s ← s + p_i
5:     if x ≤ s then
6:         return i;
7:     end if
8: end for
```

The RANDOM-ASSIGNMENT algorithm performs as well as can be achieved if inter-robot communication is prohibited and robots have no prior knowledge of the current task assignments. One direction for improvement is represented by the probabilistic threshold models [13,57,82]. After initializing via the RANDOM-ASSIGNMENT algorithm, each robot would inspect the public *task* variable of its neighbors. The further the local distribution is from optimal, the more likely the robot would change its task to rectify the imbalance. While an improvement, these algorithms still have some probability of failure for distri-

butions with small relative assignments, in small-to-medium sized populations, and in networks with low average degree. Also, these threshold models are not usable in applications where robots performing the same task need to be near each other, as many neighbors will have the same tasks, and robots will switch tasks to maintain the global distribution. The remaining three algorithms address these shortcomings by measuring properties of the global distribution, then driving the global assignment towards the desired distribution deterministically.

Summary

The RANDOM-ASSIGNMENT algorithm does not measure the configuration and does actively reduce the error. The accuracy may be fine for very large systems, but for medium-sized population of 30-200, the accuracy might be unacceptable. The remaining three algorithms are deterministic and produce perfect assignments under ideal conditions.

7.5.3 Sequential Assignment

Definition

The SEQUENTIAL-ASSIGNMENT algorithm breaks the global task assignment problem into a series of n stages, one for each robot. During each stage, all the unassigned robots run a leader election algorithm (see description in Section 3.3.2) to select the robot with the lowest ID that does not have a task assignment. This robot selects a task as a function of the current stage number. This process repeats, until all robots have selected a task. The algorithm completes each stage before moving on to the next, resulting in a long execution time, as the system must complete n leader elections sequentially. However, the communications requirements are constant and small, because each leader election requires a single broadcast tree during each stage. The memory requirements are also constant and small, with each robot only needing to store the current stage and a few timers. The algorithm requires an accurate diameter estimate to know how long to wait at each stage to be sure there is only one leader.

Algorithm

We augment the robots' state with two public shared variables, *lowID* and *stage*. The algorithm executes in n stages. In every stage, a leader election algorithm is used to identify the robot in the network with smallest ID that is currently unassigned. At the beginning of the stage, each unassigned robot initializes its *lowID* variable to its own ID. During each round, each robot finds the minimum *lowID* value amongst its neighbors. If this minimum is less that the current value of *lowID*, then *lowID* is updated to this new minimum. Since this variable is public visible, it will be broadcast to neighboring robots at the end of the round. This procedure continues throughout the stage, but the smallest ID in the network, say ID_a, is stored in every robot's *lowID* variable after no more than $diam(G)$ rounds. Each robot monitors the progress of the leader election by looking for $diam(G)$ consecutive rounds during which the value of its *lowID* variable has not changed, indicating the completion of the leader election.

Now, there are three groups of robots: those that have been assigned tasks in previous stages, those that are unassigned, but with $ID > lowID$, and the single unassigned robot with $ID = lowID$. This robot is the leader. It selects its task using the sequential bining

algorithm described above, becomes inactive, and increments its *stage* variable. Inactive robots relay messages by updating their *lowID* and *stage* variables, but do not participate in the leader election algorithm. The max of the *stage* variable is computed in the same fashion as the min of the *lowID* variable. Every robot will receive the new stage value within $diam(G)$ rounds. When each robot receives a new stage value, it starts the next stage to find the next-lowest ID in the the network. This procedure continues until the robot with highest ID in the network selects its task. At this point, there are no more unassigned robots, and the stage counter will remain constant. When no new stage has been announced for more than $diam(G)$ rounds, all robots reset their stage counters, become active, and repeat the entire process. The procedure runs continuously, to adapt to population changes and other errors. If robots are added to the population, they will get their first task assignment when they become the lowest active robot. Holes in the assignment created by robots that are removed from the population will be filled by reassigning all following robots to their new tasks.

Analysis

The SEQUENTIAL-ASSIGNMENT algorithm uses a constant amount of inter-robot communications bandwidth, as the only public shared variables are *lowID* and *stage*. Each leader election and detection takes $2diam(G)$ rounds, and there are n of them, resulting in a running time of $2ndiam(G)$. Each leader election uses $ndiam(G)$ messages, so the total number of messages passed between all robots for all the leader elections is $n^2diam(G)$.

The SEQUENTIAL-ASSIGNMENT algorithm is self-stabilizing to changes in the robot population. First, consider an unassigned robot a with unique ID ID_a that is added to the system. If ID_a is smaller than the IDs of the remaining unassigned robots, robot a will become the next leader and select its task in that stage. If the ID of a is greater than all the assigned robots and the lowest unassigned robot, then robot a will wait for its turn to become leader, as if it had been present in the system since the beginning. Next, we consider the case where robot a is removed from the network. If a is unassigned and there exists another unassigned robot b in the network with $ID_b < ID_a$, then the algorithm has not yet arrived at the stage where robot a would have selected its task, so its absence will not cause any errors. If robot a has already selected its task, then its removal will create a gap in the task assignment that will be rectified in the next loop, after all other robots have selected their tasks and the stage counter is reset.

The case in which care must be taken is if robot a is the unassigned robot with the smallest ID when it is removed. This means that robot a was active and would have been the new leader and selected its task in this stage. Neighboring robots might have propagated ID_a as the lowest ID, so other robots in the network will have stored this in their *lowID* variable. Since a is removed before it selects its task and increments the stage number, the other robots will keep $lowID = ID_a$ indefinitely, as robot a will not be present to increment the stage counter. This is a liveness error and will halt execution indefinitely. We solve this problem by adding a liveness-error check so that each robot detects if the value stored in *lowID* does not change for longer than $2diam(G)$ rounds. This is longer than the time it would take for the *stage* variable to be incremented by robot a if it were present. Each unassigned robot detects this condition independently, reinitializes their *lowID* variable to their own ID, and restarts the current stage to select a new leader. Figure 7-4 shows the full finite-state machine diagram for the SEQUENTIAL-ASSIGNMENT algorithm.

We prove correctness and running time bounds under ideal conditions of static con-

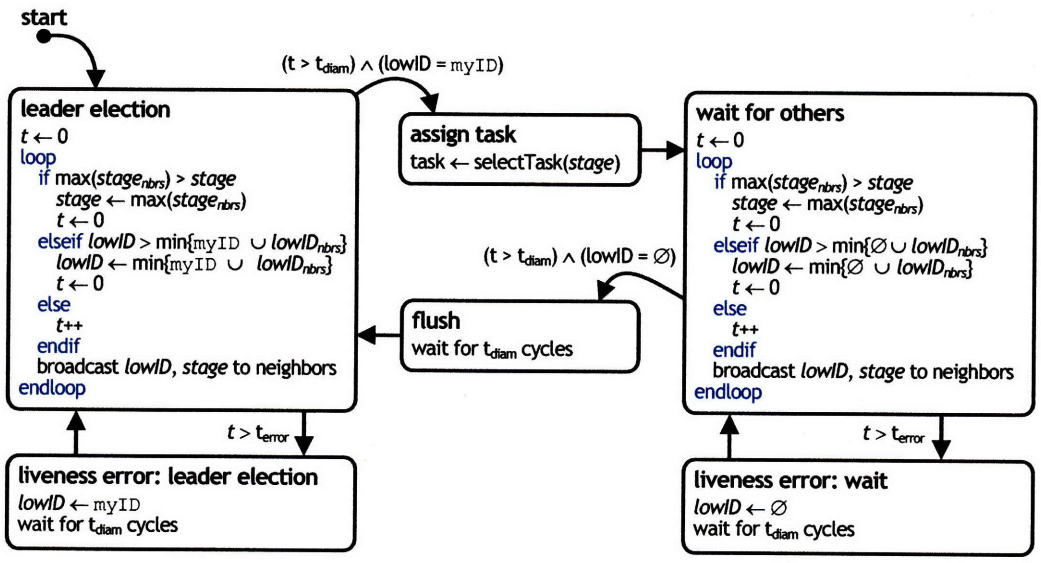


Figure 7-4: Sequential assignment algorithm finite-state machine diagram. The loop structures execute once each round.

figuration and no message loss. We allow for the addition or removal of any number of robots.

Theorem 7.5.1 *The* SEQUENTIAL-ASSIGNMENT *produces a correct assignment in* $O(ndiam(G))$ *rounds after initialization, addition or deletion of any robot.*

Proof There are three cases:

Initialization After initialization, a single leader is elected in each stage in $diam(G)$ rounds. This leader knows its stage number, and selects its task in no more than $diam(G)$ rounds after it is elected using the sequential bining algorithm from Section 7.4.2. There are at most n stages, so the total number of rounds to a correct assignment is $2n\ diam(G)$.

Addition Adding a robot to any stage has two cases: **Case 1:** the new robot has ID greater than the current leader. The leaders are selected in order of their ID, so the new robot will become the leader in a later stage, and will select its task on the stage when its ID is the lowest. The number of rounds to a correct assignment will remain $n\ diam(G)$. **Case 2:** the new robot is added and has an ID that is less than the active robot. In this case, the new robot will become the active robot, and select its task out-of-order, but the group will still produce the correct global assignment. During the next cycle through the robots, the algorithm will re-assign tasks to the robots in the correct order. The total number of rounds to a correct assignment will be $2n\ diam(G)$.

Removal There are three cases to consider when removing a robot: **Case 1:** The robot is removed after selecting its task. This will create an error in the global distribution that will be corrected after the next cycle through the robots. This could take up to $4n\ diam(G)$ rounds. **Case 2:** The robot is removed before it selects its task, and not in

the stage in which it would be the leader. In this case, the algorithm will continue as if the robot were never present, select the correct tasks for the remaining robots, and produce a correct assignment in $2n\,diam(G)$ rounds. **Case 3:** If the robot is in the process of becoming the leader and has started propagating its ID, then its removal will cause a liveness error. The algorithm will restart the stage after $2\,diam(G)$ rounds, so the total running time will be $(n+2)diam(G)$ rounds.

In all cases, the algorithm produces the correct assignment in $O(n\ diam(G))$ rounds. □

Summary

The SEQUENTIAL-ASSIGNMENT algorithm uses little communication resources, runs slowly, and determinately produces a correct assignment. The next algorithm trades off communication and running time in the opposite way.

7.5.4 Simultaneous Assignment

Definition

In the SIMULTANEOUS-ASSIGNMENT algorithm, each robot constructs a list of all the robots in the network, and selects its task based on its relative position in this list. The list is kept up-to-date with continuous message passing, allowing robots to adjust their assignments in response to population changes. The algorithm runs quickly, but requires a large amount of inter-robot communication bandwidth.

Algorithm

Each robot adds the list *messageList* to its public shared variables. This variable is a list of tuples of the form:

$$messageList_i \equiv \langle ID, active, timeStamp, timer \rangle.$$

During each round, each robot queries the message lists from its neighbors. A given robot a processes the message lists from its neighbors in the following fashion: If a message list from a neighbor contains a message from a robot that is not on a's list, this new robot is added to a's list. A message from a robot that is already on a's list is updated if the *timeStamp* field of the new message is larger than the time stamp of the message currently on a's list. The *timer* field for any message added to or updated on the list is reset to MESSAGETIMEOUT, which is a system constant.

At the end of each round, all robots decrement the *timer* field of each message on their message lists. Messages with *timer* $= 0$ are labeled as INACTIVE. Inactive messages are kept on the list to prevent messages from the same neighbor with the same *timeStamp* value from being reintroduced as new messages. Next, each robot, say robot a, adds or updates its own message to its list, creating a message composed of:

$$message_a \equiv \langle ID_a, \text{ACTIVE}, timeStamp_a, \text{MESSAGETIMEOUT} \rangle.$$

The *timeStamp* variable is a nonce that is incremented once each neighbor round, and each robot has its own *timeStamp* variable. Because the *messageLists* are public shared variables, they are broadcast to neighboring robots each round, by the one-hop communication system. This allows each robot to query the message lists of its neighbors.

Algorithm 6 DTA-SIMULTANEOUS-ASSIGNMENT($\mathbf{P}_{\text{goal}}$)

```
1: myMessage ← {MYID, ACTIVE, 0, MESSAGETIMEOUT}
2: messageList ← myMessage        ▷ messages are tuples of {ID, active, timeStamp, timer}
3: loop
4:     (wait until end of round)
5:     for all e ∈ messageList do                                  ▷ Remove old messages
6:         if e.timer = 0 then
7:             e.active ← INACTIVE
8:         else
9:             e.timer- -
10:        end if
11:    end for
12:    myMessage.timeStamp++
13:    messageList ← messageList ⋃ myMessage
14:    for all m ∈ incomingMessages do
15:        if (message is from new robot) or
16:          (m.timeStamp > (timeStamp of previous message from same robot)) then
17:            m.timer ← MESSAGETIMEOUT
18:            m.active ← ACTIVE
19:            messageList ← messageList ⋃ m                       ▷ Update or add m to msgList
20:        end if
21:    end for
22:    SIMULTANEOUSASSIGNMENTSELECTTASK(P_goal, messageList)
23:    NBRBROADCAST(messageList)
24: end loop
```

The SIMULTANEOUS-ASSIGNMENT algorithm runs continuously, once each round. The data structure that is shared is each robot's *messageList*, but it is easier to describe the algorithm in terms of messages from individual robots, say, all the messages from robot a. As each robot broadcasts its *messageList* variable to its neighbors, messages from individual robots propagate throughout the network. No more than $diam(G)$ rounds after initialization, each robot will have a complete list of all the robots in G. Also, a new robot added to the network will be known to all other robots in at most $diam(G)$ rounds. If a robot is removed, its absence will be noted within $diam(G)$ + MESSAGETIMEOUT rounds by all robots, since it will no longer be present to update its time stamp, and all of its messages will be labeled INACTIVE after MESSAGETIMEOUT rounds.

The implementation uses a bounded *timeStamp* ranging from 0-255, which adds some complexity to the removal of robots from the message lists. Because the *timeStamp* variable will wrap around after reaching a maximum value, the greater-than comparison must be replaced with a circular range of admissible and inadmissable values. It is desirable to minimize the range of inadmissable values, as a previously inactive robot will not be able to re-enter the network until its *timeStamp* increments into the admissible range. Another solution is to remove inactive robots from the *messageList* after a fixed amount of time $t > diam(G)$. This would prevent the re-introduction of old messages from robots that become inactive, but allow the inadmissable period to be small, after which inactive robots would be removed from the *messageList* variables of other robots. After this time, a previously inactive robot can re-enter the network immediately. We use the second approach in our implementation.

Analysis

The SIMULTANEOUS-ASSIGNMENT algorithm computes a correct assignment in $diam(G)$ rounds after initialization, or the addition or removal of any robots.

Lemma 7.5.2 *It takes no more than $diam(G)$ rounds for any new robot to be added to the lists of all the robots.*

Proof At initialization, a robot, say robot a, only knows of itself, and its message list contains only one entry; its own message, msg_a. During each round, each robot receives a message list from each of its neighbors. In this way, msg_a propagates one hop away from robot a. The maximum number of hops any message has to travel is the diameter of the graph, and each hop takes one round, so the total time required for m_a to reach all of the robots is $diam(G)$ rounds. □

Lemma 7.5.3 *It takes no more than $diam(G)$ + MESSAGETIMEOUT rounds from the time a robot is removed from the network to the time that robot's messages are removed from the list of every other robot.*

Proof If robot a is removed, it will no longer be present to increase the value of its *timeStamp*. Each robot combines the message lists from its neighbors with its own, keeping the message with the largest *timeStamp* from each robot. Eventually all of the robots will contain the last message from robot a, the one with the largest time stamp, and this value will remain constant on each robot. After detecting a constant time stamp for more than MESSAGETIMEOUT rounds, each robot will label this message as INACTIVE. Inactive messages are left on the list, but not published externally. The last robot to remove the message from

robot a can be no further than $diam(G)$ hops away from a's previous position. It takes MESSAGETIMEOUT rounds before the message from robot a is labeled INACTIVE. Therefore, the worst-case time for all robots to label messages from robot a inactive is $diam(G)$ + MESSAGETIMEOUT rounds. □

Lemma 7.5.4 *If a robot knows the total number of robots and its position in the ordered list of all robots, then it can pick its task optimally, within the precision allowed by the integral number of robots.*

Proof Given the goal distribution, the total number of robots, n, and a position in the list, each robot selects a task by taking its position in the list and bining it with respect to $\mathbf{P}_{\text{goal}}$, as described above. □

Theorem 7.5.5 *The* SIMULTANEOUS-ASSIGNMENT *algorithm will assign an optimum task assignment in no more than* $diam(G)$ + MESSAGETIMEOUT *rounds after initialization, the addition of any number of robots, or the removal of any number of robots.*

Proof Whenever the population has been stable for $diam(G)$+MESSAGETIMEOUT rounds, by lemma 7.5.2 and lemma 7.5.3, all of the robots in the network will have the same message list, i.e., they will have reached a consensus on their message lists. With the length of this list and its own position on the list, by lemma 7.5.4 each robot can select the correct task.

I suspect that there can be no deterministic algorithm that runs faster than $O(diam(G)$ rounds. Any algorithm that requires all the robots to have any global information will need at least at least this amount of time to propagate the information throughout the network. Any algorithm that does not use global information cannot guarantee to produce a globally correct assignment. The global communications structure is broadcast tree communications. Each robot is the source of the message about itself. As this message propagates to all other robots, robots only accept messages with newer time stamps that their current message. This enforces an acyclic propagation away from the source, but it does not build a tree because the messages do not encode hops from the source or parent in the tree. As such, this is a degenerate form of a broadcast tree, but it still has broadcast tree global communication structure, because information must always flow away from the source robots.

The communications complexity, $\mathcal{C}_{\text{SimultaneousAssignment}}$ is $O(n)$, since each robot must send one message for every robot in the network, which makes this algorithm impractical for a swarm with limited communication bandwidth. The expected total number of messages sent by all robots during convergence scales as $n^2\ diam(G)$. The algorithm collects a large about of global information on each robot, the IDs of all the other robots, which is not needed to solve the problem. Only one piece of information is needed to compute the task assignment: the robot's position in the sorted order of all robots.

Summary

The SIMULTANEOUS-ASSIGNMENT and SEQUENTIAL-ASSIGNMENT algorithms represent two extreme points in running time and communications trade-offs. It seems likely that no deterministic algorithm can assign globally correct tasks faster SIMULTANEOUS-ASSIGNMENT

which is limited by the time it takes for a message to propagate throughout the network, $O(diam(G)$ rounds. The running time of n $O(diam(G))$ rounds for SEQUENTIAL-ASSIGNMENT seems about as slow as possible while still making forward progress, although one could certainly design an arbitrarily slow protocol.

The analogous situation is true for communication, while the SEQUENTIAL-ASSIGNMENT uses the fewest possible communication resources, $C_{\text{SequentialAssignment}} = O(1)$ bits/robot/round, the SIMULTANEOUS-ASSIGNMENT uses a very large amount of communication, $C_{\text{SimultaneousAssignment}} = O(n)$ bits/robot/round. This is more than our assumptions allow, and this algorithm can only run on very small populations in our experimental setup. However, the smallest number of messages used by each algorithm before arriving at the first correct assignment scales as $O(n^2\ diam(G))$. This global constant suggests there might be a lower bound on the number of total bits required to reach any accurate deterministic assignment based on global information.

7.5.5 Tree-Based Assignment

Definition

The TREE-BASED-ASSIGNMENT algorithm promises a running time of $O(diam(G))$, comparable to SIMULTANEOUS-ASSIGNMENT but with a lower communications complexity: $C_{\text{TreeBasedAssignment}} = O(k \cdot m)$, where k is the total number of tasks and m is the maximum degree of the robot's vertices in G. The TREE-BASED-ASSIGNMENT algorithm uses a convergecast to accumulate the current global task assignment onto a single robot. This robot computes the difference between the current assignment and the desired assignment, then prepares *retasking* messages to effect the required changes across all the robots. The retasking messages are propagated down the tree, and robots change their tasks behind the "wavefront" of this message. When all the robots have processed their retasking messages, the swarm will have achieved the desired distribution. The root waits for $T = diam(G)$ rounds to allow the retasking messages to propagate all the way to the leaves. After waiting, the entire process repeats, allowing the system to respond to population changes.

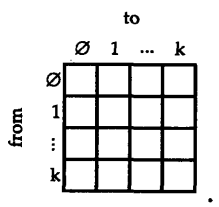
The algorithm require the routing tree must remain relatively stable over all the phases of message broadcast, summation, and retasking. A small number of communications errors can have a large affect on the accuracy of this algorithm in stationary robots, and it is highly sensitive to network topology changes in moving robots.

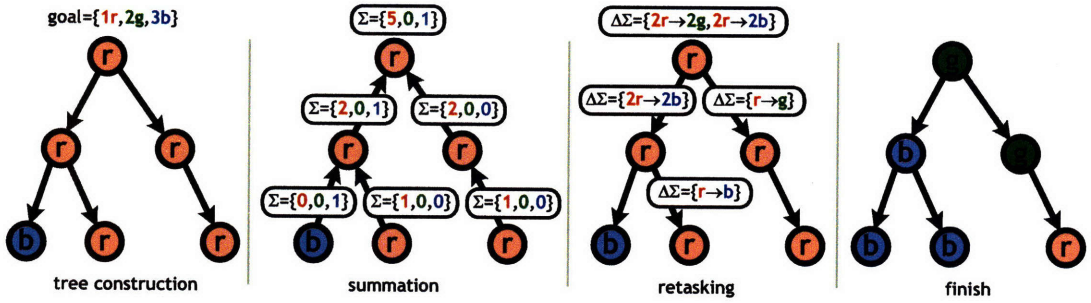
Algorithm

Each robot adds two variables to each robot's public shared variables. The first is the *subtreeAssignmentSums*, and it is a tuple with $(k+1)$ elements of the form:

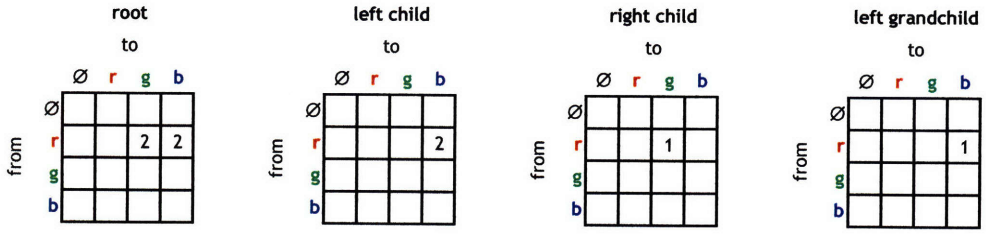
$$\langle n_{\text{unassigned}}, n_1, \ldots, n_k \rangle.$$

The second is a list of m *retasking arrays* of the form:





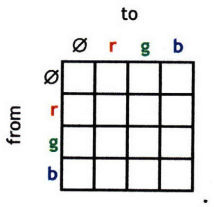
(a) The stages of algorithm execution.



(b) Retasking arrays for the retasking phase of the example above.

Figure 7-5: The TREE-BASED-ASSIGNMENT algorithm

Throughout this section, we will use an example with $k = 3$ tasks, with task names **red**, **green**, and **blue**. The form of the retasking array for our example is:



The TREE-BASED-ASSIGNMENT algorithm uses leader election to select a single robot to be the root of a broadcast tree. The algorithm execution is broken into two phases: broadcast/convergecast and retasking. Diameter estimation is running concurrently at all times, so the root always has a current diameter estimate. During this description, we will refer to the the example in Figure 7-5 to clarify the discussion.

Broadcast/Convergecast: The broadcast and convergecast algorithms run concurrently. The leader is the root of a broadcast tree, it is not important which robot is the leader, so long as there is one, we select the robot with the lowest ID. The robots use a convergecast message to tally the current task assignment. The execution is similar to the description in Section 3.3.3, except there are $k+1$ convergecasts running in parallel, one for each task and one for unassigned robots. It is helpful to describe the summation process starting at the leaves of the tree. Each leaf publishes a *subtreeAssignmentSum* tuple of zeros with a single one in the position of that robot's current assignment. For example, the blue leaf would publish: $\langle 0, 0, 0, 1 \rangle$. These variables are public, so parents query the *subtreeAssignmentSums*

of their children, sum them element by element, add their own tuple from their current assignment, then publish the results. In our example, the *subtreeAssignmentSums* of the root is $\langle 0, 5, 0, 1 \rangle$. Under ideal conditions of static network and no message loss, the root will receive a correct convergecast sum in a maximum of $d = diam(G)$ rounds after the initial broadcast tree reaches the leaves. Once the sum at the root remains stable for d rounds, the root can be certain that it has received all the sums from all the leaves. At this time, the root transitions to the retasking phase.

Retasking: At the beginning of the retasking phase, the root has the current global assignment of tasks. The sum of all the global task assignments is n, the total number of robots. The root uses the number of robots, the bining algorithm and the goal assignment, $\mathbf{P}_{\text{goal}}$, to determine the integer number of robots that should be assigned to each task. , then compare this goal assignment to the current assignment. The differences in these two assignments allows the root to compute a *retasking-array* R, a matrix in which the (i, j)-th entry is the number of robots assigned to task i that should switch to task j. The retasking array for the root in our example is shown on the left hand side of Figure 7-5b. The root changes tasks according to the array, updates R, and then calculates and transmits *subtree retasking arrays*, $R_1, \ldots, R_m$, one to each of its m children. Since each robot knows the current assignments each of its children reported for their subtree, the robot can iterate through the children and give them reassignments based on their current subtree assignments. The retasking arrays for the children of the root in our example is shown in the two middle arrays of Figure 7-5b. Essentially, the robot is transferring assignments from R to R_i, updating R, and repeating until retasking assignments have been handed out to children. This process repeats at each level of the tree. The retasking arrays use the broadcast tree to reach the entire network. After $diam(G)$ rounds, they reach the leaves.

The root uses its diameter estimate to know how long to wait for the retasking to complete. After $O(diam(G))$ rounds, it starts the next broadcast/convergecast phase. The start of the next phase signals the robots to accept their new task assignments, and begin convergecasting the updated assignments towards the root.

Analysis

Each of the two phases takes $O(diam(G))$ rounds, so the total running time of the TREE-BASED-ASSIGNMENT algorithm is $O(diam(G))$, comparable to that of the SIMULTANEOUS-ASSIGNMENT algorithm. The communication complexity scales as $O(k \cdot m)$. The *subtree-AssignmentSums* variable is of size $k+1$. Each non-leaf robot publishes retasking messages to each of its children, of which there are $O(m)$. This communications usage is better than SIMULTANEOUS-ASSIGNMENTWhile $k \cdot m$ is not small, it does not grow as a function of n. If bandwidth is limited, retasking massages can be sent to a fixed number of children during each phase, selecting different children randomly each phase. In a stationary system of robots, this will update different subtrees on different phases, eventually updating the entire population. This will extend the total expected running time by the $\frac{1}{s}$, where s if the percentage of children that are communicated with each round. Our experimental system has a fixed maximum number of neighbors of $m_{max} = 16$, but this is too many retasking messages for the available bandwidth, so we apply the second approach in our implementation and use a maximum of four retasking arrays.

The total number of messages passed in one full set of phases of the algorithm scales as $O(m\ n\ k\ diam(G))$, which is one factor less in n than SEQUENTIAL-ASSIGNMENT and

SIMULTANEOUS-ASSIGNMENTThis makes sense, as this algorithm only accumulates global information on a single robot, the root.

We now prove key properties of the algorithm.

Lemma 7.5.6 *During the broadcast/summation phase, it takes no more than $2diam(G)$ from initialization, addition, or deletion of any robot before the root of the broadcast message has an accurate estimate of the current task assignment.*

Proof We can break this into 3 cases:

1. **Initialization:** It takes $diam(G)$ time to construct a broadcast tree, $diam(G)$ to convergecast the current assignments back to the root of the tree, and $diam(G)$ for these assignments to remain stable long enough for the root to switch to the retasking phase.

2. **Addition of robots:** The new robot will receive a message from its parent on the broadcast tree within one round. It will than take $diam(G)$ rounds to convergecast the new assignments back to the root of the tree, and $diam(G)$ for these assignments to remain stable long enough for the root to switch to the retasking phase. The total time is $2diam(G) + 1$ rounds.

3. **Removal of robots:** The parent of the removed robot will notice the removal after p (p is the neighbor persistence constant. See Section 3.3.2) rounds . It will than take $diam(G)$ rounds to convergecast the updated assignment sums back to the root of the tree, and $diam(G)$ for these assignments to remain stable long enough for the root to switch to the retasking phase. The total time is $2diam(G) + p$ rounds.

The worst case time is initialization, which is $3diam(G)$ rounds. □

Lemma 7.5.7 *During the retasking phase, if no robots are removed or added, it takes no more than $diam(G)$ rounds for each robot to receive their new task assignment and switch to the correct task.*

Proof The broadcast of the retasking arrays propagates at the same speed as the broadcast tree. Therefore, it takes $diam(G)$ rounds to reach every robot in the network. □

Theorem 7.5.8 *The* TREE-BASED-ASSIGNMENT *algorithm produces a correct assignment in no more than $O(diam(G))$ rounds from initialization, addition or removal of any robot.*

Proof Lemma 7.5.6 bounds the time for the broadcast/summation phase to $3diam(G)$ rounds, and Lemma 7.5.6 bounds the retasking phase to $diam(G)$ rounds, so an entire cycle of execution takes $4diam(G)$ rounds, but this does not tolerate additions or removals during the retasking phase. If robots are added during the the retasking phase, they might receive a new task, but their addition will make the count the root used to construct its retasking array incorrect, resulting in not enough retasking commands to account for the larger population. If a robot is removed during retasking, it will disconnect a (potentially large) subtree from the root, and many robots will not receive their retasking commands.

In both cases, the solution is to wait for the next cycle to begin. The root will receive the convergecast with the correct population, and distributed new retasking arrays. The worst-case running time for all this is $8diam(G)$ rounds, depending on when the robots were added or removed. This is still $O(diamG)$ execution time.

Algorithm	Running Time	Bandwidth per Robot	Total Bandwidth	Error Variance
RANDOM-ASSIGNMENT	$O(1)$	0	0	$O(1/n)$
SEQUENTIAL-ASSIGNMENT	$O(n\ diam(G))$	$O(1)$	$O(n^2\ diam(G))$	0
SIMULTANEOUS-ASSIGNMENT	$O(diam(G))$	$O(n)$	$O(n^2\ diam(G))$	0
TREE-BASED-ASSIGNMENT	$O(diam(G))$	$O(k\ m)$	$O(n\ m\ k\ diam(G))$	0

Table 7.1: Comparison of the asymptotic theoretical upper bounds on the four algorithms' running time, communications usage, and accuracy under ideal conditions of static configurations and no message loss. The notation n, m, k, and G denote the total number of robots in the network, the maximum degree of the network, the number of tasks in the goal distribution, and the network graph, respectively. Running time is defined as the number of rounds between stabilization of the robot population and stabilization of the final task distribution. The per-robot per-round communications rate is defined as the number of messages sent by a single robot in a single round. The total communications burden on the network is defined to be the sum over all robots of the per-round per-robot communications rate until the algorithm converges. Accuracy is measured by the variance of the theoretical error distribution between final stabilized state and actual target distribution.

Summary

The TREE-BASED-ASSIGNMENT algorithm is a compromise between the SEQUENTIAL-ASSIGNMENT and the SIMULTANEOUS-ASSIGNMENT in terms of communications usage and running time. Its downside is the complexity of the global communications structure it employs.

7.6 Algorithm Comparison

Each of the algorithms represents a different trade-off in resource usage. Table 7.6 shows the differences in theoretical performance of the four algorithms in this section. The RANDOM-ASSIGNMENT algorithm uses no communication , but doesn't provide any accuracy guarantees, just expected performance. All three of the deterministic algorithms drive the error to zero. The SEQUENTIAL-ASSIGNMENT and the SIMULTANEOUS-ASSIGNMENT algorithms represent two extremes of the communications/running time trade-off. Although they both send the same total number of messages to reach the first correct assignment, the SIMULTANEOUS-ASSIGNMENT algorithm sends them all at once, executing is what seems to be the shortest time possible for a deterministic algorithm that uses global information. By comparison, SEQUENTIAL-ASSIGNMENT algorithm sends very few message each round, and runs in a very long time, perhaps the longest time possible that is always making forward progress. The TREE-BASED-ASSIGNMENT centralizes the computation onto a single robot, reducing the total number of messages that need to be sent before the first stable assignment, but requiring a stable network structure with reliable communications links. While attractive on paper, the implementation is fairly complex and the broadcast + convergecast + rebroadcast global communication structure will not fare well in dynamic networks.

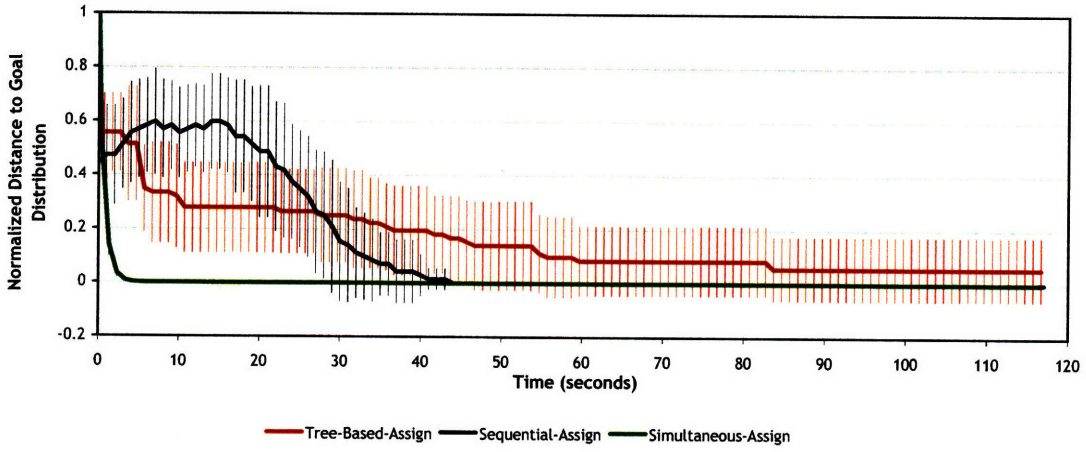


Figure 7-6: The convergence properties were measured by running each algorithm 8 times on a group of 18 robots, and measuring the normalized error as a function of time. In all runs, the robots started from an random initial assignment, and converged towards $(\frac{1}{6}, \frac{1}{3}, \frac{1}{2})$. The SIMULTANEOUS-ASSIGNMENT and SEQUENTIAL-ASSIGNMENT algorithms' temporal performance was close to the theoretical limits, and both converged to the correct distribution in all runs with no error. The TREE-BASED-ASSIGNMENT algorithm struggled with communication errors which extended its running time well beyond the theory, but it still converged close to the desired distribution.

7.7 Experimental Results

Each SwarmBot is adorned with large red, green, and blue "Behavior LEDs" and has a MIDI audio system. We use these lights and sounds to monitor the internal state of the robots. In particular, the task a robot has selected is represented by illuminating one of these LEDs, so we limited our experiments to three task groups – red, green and blue – to allow the use of a standard video camera for data collection.

We conducted three sets of experiments to measure each algorithm's assignment accuracy and convergence time, running time as a function of total number of robots, and stability to external disturbances.

7.7.1 Convergence and Accuracy

The first experiment was designed to measure convergence time and accuracy. Convergence error is defined as the distance from the current distribution vector and a fixed goal distribution $\mathbf{P}_{\text{goal}} = (\frac{1}{6}, \frac{1}{3}, \frac{1}{2})$: $e = ||T(t) - \mathbf{P}_{\text{goal}}||_2$. The results are shown in Figure 7-6. All three algorithms were highly accurate, with SIMULTANEOUS-ASSIGNMENT and SEQUENTIAL-ASSIGNMENT producing final assignments with no errors. The SIMULTANEOUS-ASSIGNMENT algorithm converged the fastest, but SEQUENTIAL-ASSIGNMENT outperformed TREE-BASED-ASSIGNMENT on average, ignoring the theoretical results that predict otherwise. The SEQUENTIAL-ASSIGNMENT algorithm uses little communications, and ran exceptionally well on the Swarm. But, even for this small swarm, the average convergence time was 53 seconds, making this algorithm impractical for large swarms. The

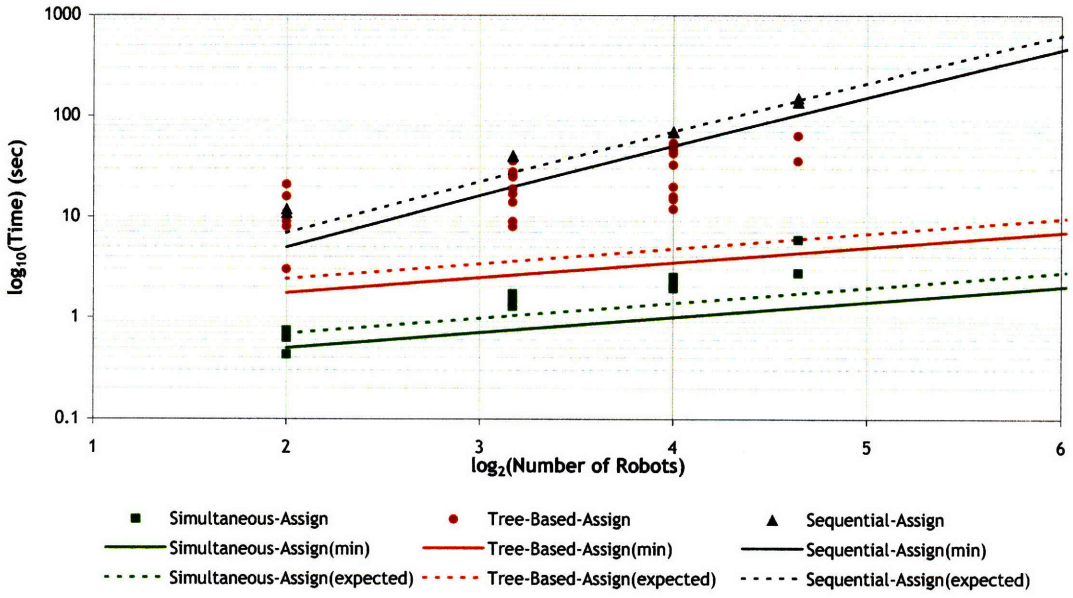


Figure 7-7: The running time of all three algorithms scale with the total number of robots in the network. This graph shows time plotted against the number of robots on log-log axes. Configurations containing between 4 and 25 robots were tested. Solid lines show the predicted running time with no communications errors, dashed lines show the expected running time with typical errors. Dots are data points from individual experiments. The SEQUENTIAL-ASSIGNMENT and SIMULTANEOUS-ASSIGNMENT algorithms performance was close to the expected running time, while the TREE-BASED-ASSIGNMENT algorithm was far from expected. This was caused by incorrect summation information propagating back to the root of the broadcast tree, making the subsequent retasking commands incorrect.

TREE-BASED-ASSIGNMENT algorithm relies on broadcast trees to propagate partial sums of the current distribution back to the root robot. Messages dropped during this process were common, which caused incorrect summations, leading to an incorrect distribution accumulating at the root, and then incorrect retasking arrays sent back down the tree. Some of the experimental runs showed the promise of the TREE-BASED-ASSIGNMENT algorithm, with the correct assignment being achieved in two retasking cycles (21 seconds), only twice the minimum running time. There are techniques in the literature to help stabilize the tree computations (taking the max over windows of time, encoding packet routing history to deal with topology changes [81], etc.), some of which might bring the convergence time closer to the theoretical limits. It is interesting to note that SIMULTANEOUS-ASSIGNMENT and TREE-BASED-ASSIGNMENT both measure the current distribution and use feedback to converge monotonically toward the goal, while SEQUENTIAL-ASSIGNMENT makes uninformed task assignments at each round, causing the global error to increase while running.

7.7.2 Running Time

The second experiment measured the convergence time as a function of the number of robots in the network. We progressed from 4 to 25 robots, which is close to the bandwidth limit for SIMULTANEOUS-ASSIGNMENT. We arranged the robots so that the diameter of the network was known *a priori* and the minimum running time could be computed directly. For each group size, several runs from a random initial distribution to an assignment of $\mathbf{P}_{\text{goal}} = (\frac{1}{6}, \frac{1}{3}, \frac{1}{2})$ were timed. The round of 250 ms and the algorithm design were used to compute a lower bound on the running time, but actual running time depended on communications errors, as lost messages impeded performance. This can be modeled by extending the round and calculating expected running times. The expected round is $\sim$ 37% longer than the minimum [73]. The results are shown in Figure 7-7.

The SIMULTANEOUS-ASSIGNMENT algorithm performed well, but as the communications burden increased, the measured running times moved further from the expected running times, possibly because the communications usage was outside of the usage patterns used to characterize the system. The SEQUENTIAL-ASSIGNMENT algorithm ran perfectly each time, generating data with almost no variance. Again, the TREE-BASED-ASSIGNMENT algorithm ran slower than expected, with a large variance in convergence time. The source of errors was the same as the previous experiment - incorrect sums caused by fragile broadcast trees.

7.7.3 Disturbance Rejection and Self-Stabilization

The third set of experiments measured the disturbance rejection properties of the three algorithms. Four disturbances were introduced in sequence: 1. Initial transition from $\mathbf{P}_{\text{goal}} = \textit{null}$ to $\mathbf{P}_{\text{goal}} = (1,0,0)$, 2. transition from $\mathbf{P}_{\text{goal}} = (1,0,0)$ to $\mathbf{P}_{\text{goal}} = (0, \frac{1}{2}, \frac{1}{2})$, 3. transition from $n = 24$ to $n = 12$ while keeping $\mathbf{P}_{\text{goal}}$ constant, 4. transition from $n = 12$ to $n = 24$ while keeping $\mathbf{P}_{\text{goal}}$ constant. A centralized radio network was used to broadcast the new distribution and population commands to a group of 24 robots. Figure 7-8 shows the results, with grey lines indicating when each of the disturbances occurred.

The SIMULTANEOUS-ASSIGNMENT algorithm should have easily handled all of these disturbances. Unfortunately, problems with the robots' radios corrupted many of the data sets. Attempts were made to separate errors caused by the experimental setup from errors caused by inter-robot communications, but unexplained variances are still present. The distribution change at $t = 5$ occurs very quickly, because each robot has already compiled the list of all the other robots, and can select a new task in $O(1)$ time. Removing half of the robots produced a plateau in the error from $t = 10$ to $t = 12.5$ because the time stamp in each $RobotID$ message must become invalid before that message can be removed from a list of robots. This is in contrast to the next disturbance at $t = 20$ when the 12 robots are turned back on. The algorithm adds robots as new messages arrive, quickly driving the error towards zero.

Because the SEQUENTIAL-ASSIGNMENT algorithm does not measure the current distribution, its worst case response to each disturbance is the same: $2\,n\,diam(G)$. The initial assignment and the distribution change show that SEQUENTIAL-ASSIGNMENT only changes the task of one robot at a time. The small error after the population change from 24 to 12 robots is misleading, as removing random robots did not have a large effect on the global distribution, but caused SEQUENTIAL-ASSIGNMENT to change the tasks on many of the remaining robots.

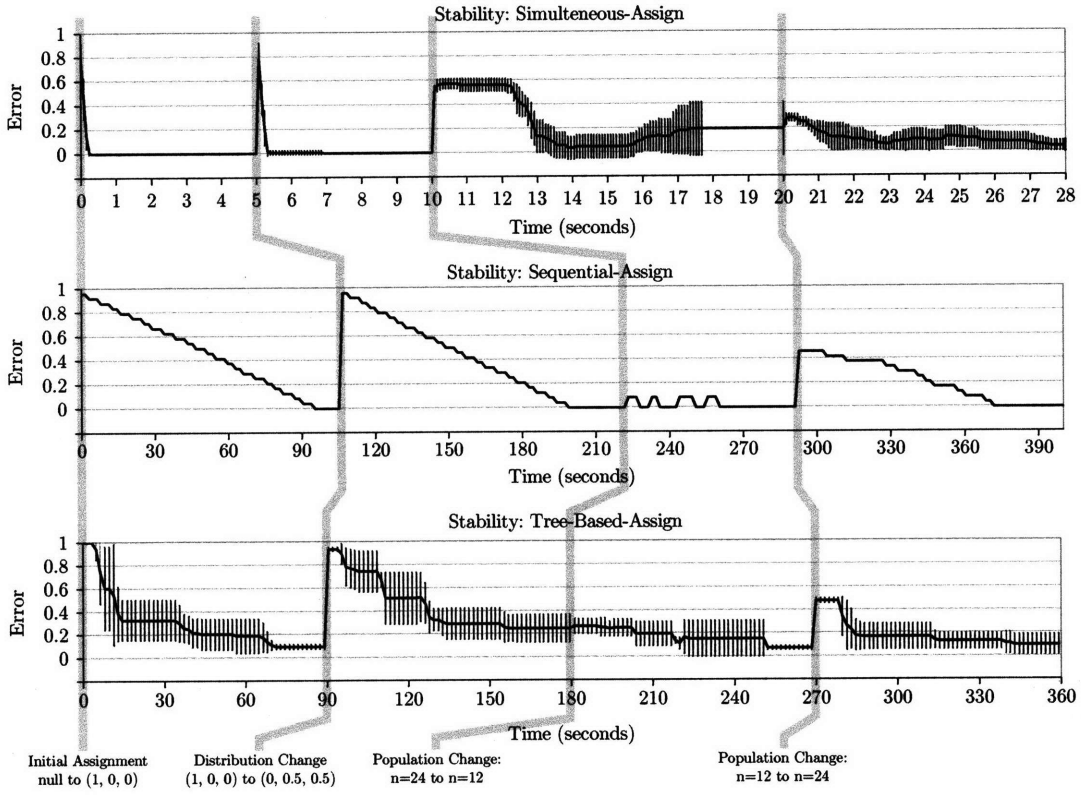


Figure 7-8: All of the algorithms are self-stabilizing when subject to external disturbances. This trio of graphs displays the recovery of the algorithms when subject to four different external disturbances; the initial convergence, a goal distribution change, removal of half of the robots, and replacement of the removed robots. Normalized error is plotted against time in all graphs, but the absolute times in each graph are different. Eight runs of SIMULTANEOUS-ASSIGNMENT produced the data on top. The algorithm performed well, but the data is partially corrupted by errors in our experimental setup. Some of this can be seen after t = 16, as the error begins to rise, even though there is no disturbance (compare with Fig. 7-6). The plot of SEQUENTIAL-ASSIGNMENT shows the algorithm's steady progress. Multiple runs produced identical results, so only one example is shown here. The magnitude of the error after the first population change is small, but many robots needed to switch tasks, which can be very disruptive. The TREE-BASED-ASSIGNMENT algorithm was hindered by the same experimental setup as SIMULTANEOUS-ASSIGNMENT as well as the convergecast summation errors mentioned in the text. It converged accurately, but not rapidly.

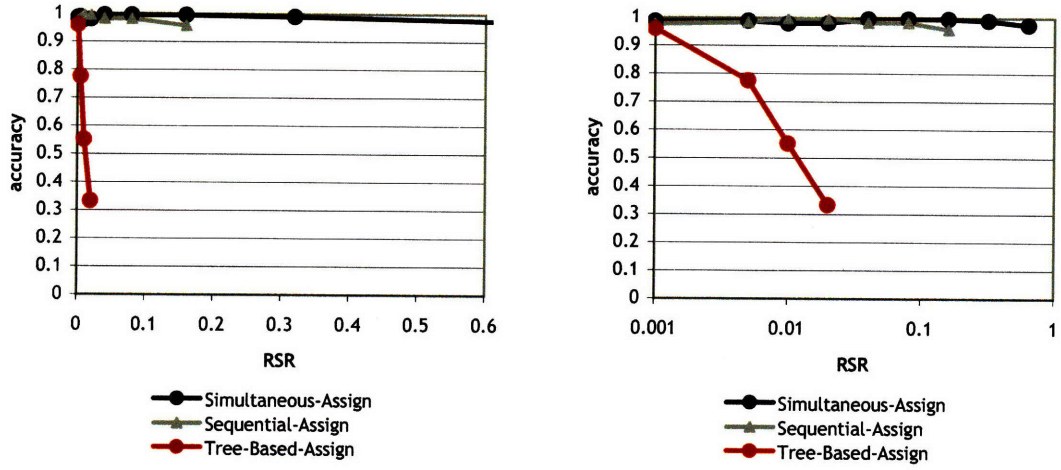


Figure 7-9: The accuracy of the three deterministic dynamic task assignment algorithms. The SEQUENTIAL-ASSIGNMENT and SIMULTANEOUS-ASSIGNMENT algorithms both essentially make a list of neighbors, and don't use communication for geometry or timely message delivery. Their performance is largely unaffected by the RSR. The TREE-BASED-ASSIGNMENT algorithm uses a broadcast = convergecast + rebroadcast communication structure, and its accuracy degrades sharply as RSR increases. The algorithm was not able to operate properly at RSRs greater than 0.02. (The same data is shown on both plots. left: linear x-axis, right: log x-axis)

The TREE-BASED-ASSIGNMENT algorithm was hindered by the same experimental setup as SIMULTANEOUS-ASSIGNMENT as well as the convergecast summation errors mentioned earlier. As before, there was a large variance on convergence time, pointing to room for improvement.

7.7.4 Accuracy and Robot Speed Ratio

In order to measure the accuracy of these algorithms at varying robot speed ratios, we had to modify the experimental protocol. Because the robots store the previously selected task once an assignment has been made, the assignments do not change often once they have been initially determined. The protocol was modified to start each trial with all robots set to an unassigned task. We then sent the goal distribution to all the robots via the radio, and measured the maximum accuracy achieved within a fixed time interval of three times the convergence time for the static configuration, or two cycles through the algorithm, whichever was longer.

The three deterministic algorithms use two different underlying techniques for gathering global information and making assignments. The SEQUENTIAL-ASSIGNMENT and SIMULTANEOUS-ASSIGNMENT both effectively make a list of all the robots in the network. SIMULTANEOUS-ASSIGNMENT does this explicitly, and each robot in the the SEQUENTIAL-ASSIGNMENT has relayed messages from all of the other robots by the end of a cycle, so it has access to the same information. This process of making a list does not rely on any network geometry, or even timely arrival of messages. In fact, because the robots are in a finite

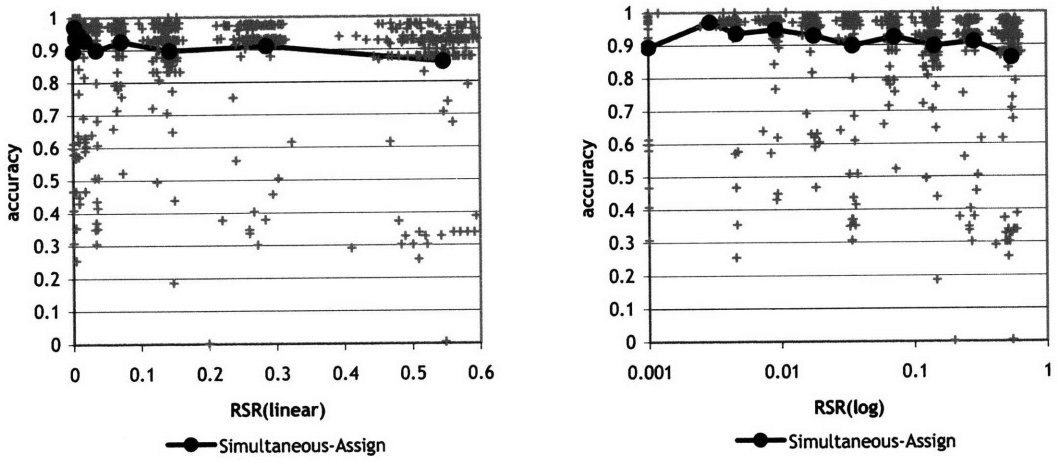


Figure 7-10: The SIMULTANEOUS-ASSIGNMENT algorithm converged rapidly enough for our standard experimental protocol. This accuracy plot was taken while changing goal distributions and adding and removing robots.

environment, each robot eventually must receive a message from every other robot. This causes both algorithms to have near-perfect accuracy at all RSRs, as shown in Figure 7-9. However, the TREE-BASED-ASSIGNMENT has a broadcast + convergecast + rebroadcast global communication structure, and requires a stable network for proper execution. In these experiments, the algorithm did not operate properly at a RSR greater than 0.02, as the root robot never received a stable measurement of the current task assignment.

The SIMULTANEOUS-ASSIGNMENT algorithm converges rapidly enough for us to use an experimental setup similar to the other chapters. At all RSRs, we changed the distribution and added and removed robots. The results of this experiment are shown in Figure 7-10. There is a very slight downward trend in the average accuracy at high RSRs, but the algorithm performs much better than any other algorithm tested at these speed ratios.

7.8 Summary

All four task assignment algorithms are doing the same underlying operation: selecting the best integer representation of a real-valued task assignment vector. We call this bining, and it might explain why the total number of messages for the SIMULTANEOUS-ASSIGNMENT and SEQUENTIAL-ASSIGNMENT algorithms are the same, both algorithms have to accumulate enough information on all the robots to run the bining algorithm on each one. It also explains why the tree assign uses less communications: it only needs to accumulate this information on a single robot, but pays a communications cost to get it there and back out. It also points towards an algorithm in between simultaneous and sequential where the robots can turn a knob to trade-off speed and bandwidth, perhaps even at run-time in response to measured network conditions.

The four solutions presented here exhibit different scaling properties and communications requirements. All four could find a place in the distributed algorithm designer's toolbox for practical dynamic task assignment. A useful enhancement would be the ability

to specify absolute requirements on the minimum or maximum number of robots assigned to a task. One of the most intriguing research directions suggested by these algorithms is exploration of resource trade-offs that are minimized by the most efficient algorithms, and are bounded below by some "conserved quantity" associated with the dynamic task assignment problem. Another important question is to understand how to combine results from this work with approaches to locally determining the optimal task distribution. Future work could follow up on this theoretical question, in addition to implementing and optimizing specific solutions.

Chapter 8

Conclusions and Future Work

8.1 Summary of Results

Figure 8-1 shows the accuracies for all the algorithms presented in this work, normalized to their performance in a static configuration. The lines are color-coded to reflect their global communication structure. We list these communications structures from least complex to most complex:

- **Black:** One-hop communications.
- **Grey:** Global communications flood, no geometry.
- **Blue:** Broadcast Tree Communications.
- **Orange:** Convergecast Communications.
- **Red:** Convergecast + Re-Broadcast Communications.
- **Green:** Broadcast Tree Communications + Physical Routing

The accuracy metrics are not designed to facilitate quantitative comparisons between different algorithms, so we limit our comments to qualitative observations. In general, the accuracy of algorithms with more complex global communication structure degrades faster as the robot speed ratio increases.

The notable exception are the two dynamic task assignment algorithms: DTA-Simultaneous-Assign and DTA-Sequential-Assign. These algorithms do not use any network geometry, or require messages to be delivered in a timely fashion. They both essentially collect all the robot IDs in the network into a list on each robot, then use that list to make assignment decisions. So any single robot will eventually receive all of the messages, and be able to make the correct task assignment.

The green line is the accuracy for a navigation algorithm. This is the only algorithm that requires motion that described in this work, and it is included because it is a fundamental building block for many other algorithms. Its performance depends directly on the geometric accuracy of the broadcast tree it uses for guidance. When the geometry of the network is poorly correlated with the actual geometry of the world, the algorithm fails completely.

In practice, I have found a RSR of 0.02 produces acceptable accuracy in a mobile system. It is important for the application designer to know what kind of accuracy to expect from

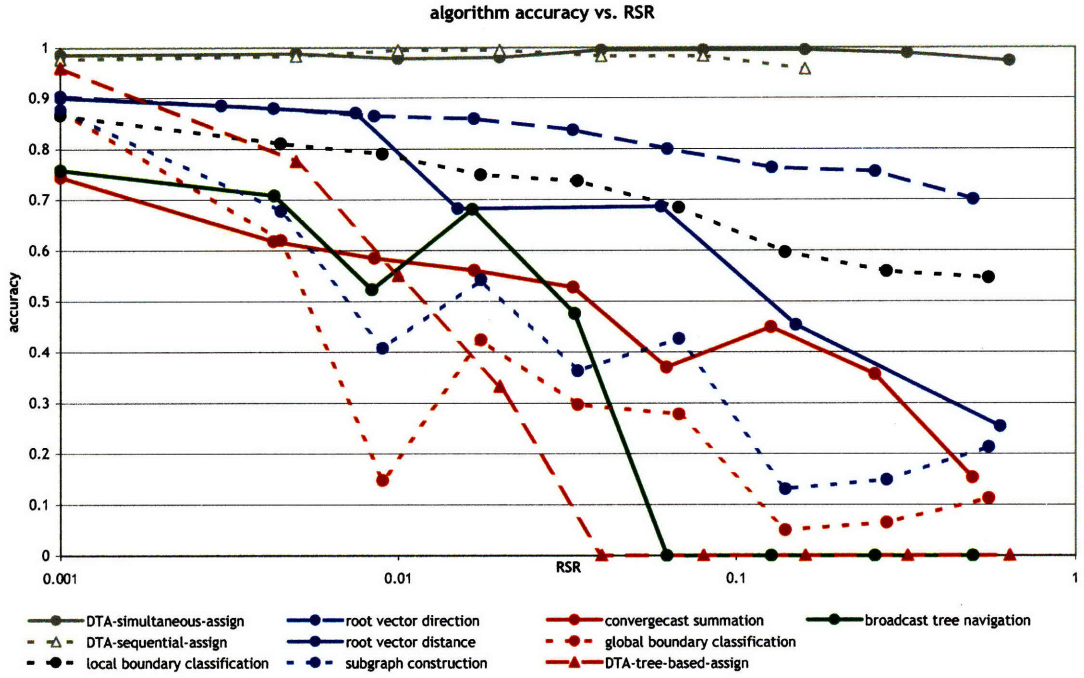


Figure 8-1: Accuracy data from all the algorithms presented in this work. See text for discussion.

an algorithm in a particular environment, and to understand what they can trade off to achieve a desired accuracy if there are system constraints. This allows the application designer to design an application around the algorithms needed to perform a task, specify a required accuracy for execution, then look up the maximum RSR that can reliable produce that accuracy. Given a maximum RSR, the designer can use the RSR equations from chapter 4 to trade off between basic system parameters, such as communications bandwidth for robot speed.

8.2 Limitations

8.2.1 Accuracy Metrics

Many of the accuracy metrics we rely on for evaluating algorithm performance are subjective and application-dependent. For example, the accuracy of a basic behavior like dispersion can be application-dependant:

Dispersion for Exploration and Searching: We want rapid, efficient motion from a single source. Want to maximize coverage to make sure the robots have explored all areas of the environment

Dispersion for Sensing: We want long-term evenly distributed coverage. Rapid dispersion into the environment is a secondary design objective.

Dispersion for Mobile Tracking: We want flexible coverage, so robots can reposition themselves to follow targets or acquire better sensor measurements of them

Dispersion for Network Connectivity: We want a robust topology with large min-cut, and a network that is flexible and allows communication loads to be balanced.

No two application will require quite the same dispersion. In general each algorithm will require a custom metric, and comparisons across accuracy metrics as defined here will be limited. Hopefully, comparisons using the same metric across different platforms will be meaningful.

In this work, a more troublesome problem was the definition of the worst possible performance. Some algorithms, like broadcast tree navigation, have a well-defined worst possible outcome: the navigating robot does not get to the root. Other algorithms produce a random distribution that is not correlated to local network geometry at high RSRs, such as the local boundary classification algorithm. However, these random distributions can be affected by the environment and experimental setup. In the case of the boundary detection, the ratio of environment perimeter to environment area is the expected worst-case performance. Note that this is not a function of the algorithm, but of the environment. It is unclear how to address this problem in general. We desire accuracy metrics that are tied to algorithm performance, not the environment or particular robot hardware.

8.2.2 Robot Speed Ratio

We use the maximum propagation message speed away from the source as the foundation of the robot speed ratio. A robot moving away from the source at a speed greater than the message speed will be effectively disconnected from the network. However, this is a conservative bound on the maximum robot speed, because it only considers the component of a robot's velocity that projects onto the vector away from the root robot. Figure 8-2 illustrates the root vector, and the velocity of a given robot, a. A more accurate estimate might be:

$$s_{\text{accurate}} = v_{\text{robot}} \cdot \overrightarrow{-R},$$

this eliminates the tangential component of the robot's velocity. However, the tangential velocity component does change network connectivity, just not away from the source.

Perhaps a better estimate of robot mobility is to measure the rate of change in network topology. Corsen [21] measures topology changes per unit time, others measure the expected amount of time until one-half of a nodes neighbors change. This would provide a direction-independent measurement of topology changes, but will not be as directly connected to the physical robot motion, or capture the notion of message speed directly. Evaluation of different network-based speed measures would be an interesting problem for future work.

8.2.3 Heterogeneous Hardware

We assume that all the robots are homogeneous. Many practical applications will require heterogeneous robots, with different mobility, sensing, computation, and communication resources. Many of the system parameters that we assume are constant across the entire group will only be constant across each class of robot in the group. It is unclear how to directly extend the RSR analysis into this domain, but it is an important question for future work.

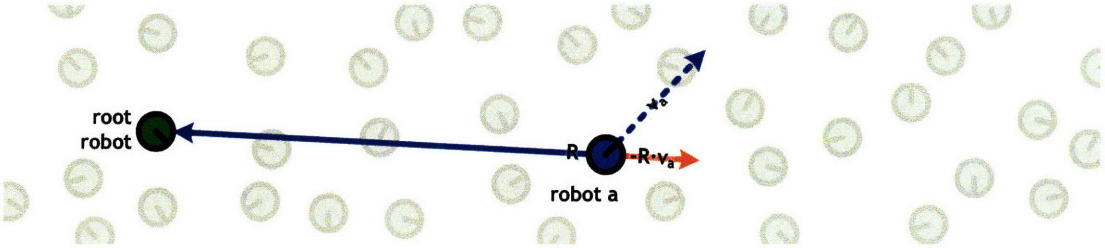


Figure 8-2: The robot speed ration as used in this work is a conservative estimate. Intuitively, the component of robot a's velocity that would seem to affect its performance the most is the component that is directly away from the root. This is the component that is limited from above by the speed of messages propagating away from the root. However, the tangential component of robot a's velocity will change its network connectivity, and affect the accuracy of its local network geometry. We leave determining the distinct effects of these two components to future work.

8.3 Future Work

There are several areas for future exploration pointed to by this work.

8.3.1 Cross-Platform Experimentation

Every effort has been taken to define the robot speed ratio and accuracy metrics based on fundamental properties of all multi-robot systems: robot speed, communications bandwidth, communication range, among others. However, it is easy for the particulars of the SwarmBot hardware platform to creep into the work, limiting the application of the ideas to physical systems very similar to our own. This will prevent the work from being a design tool for new systems, and hinder comparisons across disparate hardware platforms.

The best solution to this problem is to use the metrics to evaluate the performance of a variety of algorithms on a variety of physical systems. Until we can build a new physical system to experiment on, we will have to hope the metrics and robot speed ratio analysis are sufficiently attractive to other members of the community so that they will use them to evaluate the performance of their own algorithms and systems.

8.3.2 A Conservation Law?

The most interesting trade-offs suggested by this work are those between robot speed, communications bandwidth, and algorithm accuracy. This suggests there is some global quantity Q:

$$Q = s_{\text{robot}} \cdot B \cdot \mathcal{A}_{\text{A}}),$$

that is the product of these three key system parameters. It is not clear what exactly this quantity is, or if the units expressed here, $\frac{\text{bits·meters}}{\text{second}}$ are reasonable. This is an exciting area for future work.

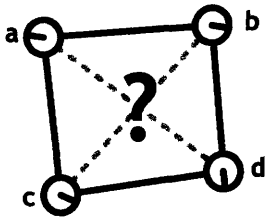


Figure 8-3: Geometric consensus in multi-robot systems can be difficult. In order to compute a globally consistent Delaunay triangulation of this network, all four robots must agree on which interior edge to keep. However, since the robots measure the positions of their neighbors locally with independent error, the four robots might make different decisions about which internal edge to keep.

8.3.3 Self-Stabilization Rate vs. Error Rate

It seems that the accuracy of an algorithm in a dynamic network is governed by two rates: The first is the *self-stabilization rate* of the algorithm. This is a measure of how quickly an algorithm can return to the desired goal configuration after a disturbance or population change. This is determined by the global communication structure required by the algorithm, the local bandwidth available on the robots, the communications complexity of the algorithm, and the diameter of the network, among others quantities.

In a dynamic network, this self-stabilization is being opposed by something we loosely describe as a *network error rate*. These errors are from changes in the network topology, communication errors, robot failures and population changes.

Perhaps another way to evaluate algorithm performance is to use more of a control-theoretic notion of disturbance rejection and steady-state error analysis. If you know the two rates in question, you can determine the system response to a variety of inputs. However, it is unclear what the units of these rates are. The running time of the algorithms in this work is specified in rounds of computation *after* some disturbance, such as a network change or population change. Given a continuous rate of disturbances, some algorithms, such as the sequential task assignment algorithm, can be made to never converge, or even make forward progress, even with a relatively low rate of disturbances. Guibas work on "Kinetic Data Structures" [6,75] is a class of algorithms that can make progress in changing topologies. What remains to be done is connect algorithm progress, network changes, and robot mobility.

8.3.4 Geometric Consensus

Issues of local coordinate systems and geometric consensus have appeared repeatedly in this work. For example, consider the four robots in Figure 8-3. They are in general position and wish to compute the Delaunay triangulation to determine which dotted edge to retain in the global graph. However, because of local sensing errors, each robot could make a different decision. If we construct a global triangulated graph from the union of each robot's local triangulated graphs, it might contain one of the two internal edges, or both of them, or neither of them. in the last two cases, the global graph formed by the union of the local triangulations is not triangulated. In order to reach a globally consistent graph, this group of four robots will need to reach a consensus, and this means that some robots might have to accept a global graph structure that is inconsistent with their local pose

measurements.

Chapter 2 describes four different classes of coordinate systems, and the main challenge of the cyclic-shape algorithm in chapter 6 was designing an algorithm that could work when there was not geometric consensus between neighbors. It is possible to produce a noisy global coordinate system from noisy local coordinates. Section2.5.1 describes techniques in the literature. It is unclear if these techniques will scale to highly mobile systems. Some of the trade-offs between them are the average node degree, the communications required, which will affect the speed of the updates.

A possible direction is to design nimble, but inaccurate global coordinate systems. If the robots' estimate of their global coordinates is incorrect, but globally consistent, a algorithm may still execute with reasonable accuracy. For example, the boundary detection algorithm would be able to form a contiguous boundary, bit all of the boundary robots might not be on the actual boundary of a dynamic network. If the robots stopped moving, the global coordinates would be able to "catch up" to the actual positions. Essentially, we are trading one type of error for another, but allowing potentially many more algorithms to be used on multi-robot systems with noisy local coordinates that are designed for systems with noisy global coordinates without modification.

Another idea is to use local "consensus overlays", where each robot is the center of a local region of geometric consensus. Each robot would enforce an ego-centric geometric consensus among its neighbors a fixed number of hops, k, away from it in the network. These regions of consensus would overlap, but would be independent. A distributed algorithm running on a robot would make it's decisions based on that robot's consensus region, and would be sure of proper operation in the k-hop neighborhood around the robot. However, this could be complicated to implement, and would require $O(m^k)$ communications per robot.

8.4 Conclusions

The main result of this work is the definition of a dimensionless measure of robot speed designed for multi-robot systems that rely on multi-hop communications. Called the robot speed ratio, I propose that the higher this speed is, the more network topology changes, and network topology changes are a significant source of distributed algorithm errors. I have demonstrated that algorithm accuracy decreases as the RSR increases in extensive empirical evaluation of ten distributed algorithms for multi-robot systems.

The foundation of this work is the definition of a set of common requirements of multi-robot applications, the development of a computational model that captures the physical nature of algorithm execution on these systems, and allows rigorous analysis of algorithm performance, while abstracting away enough complexity to be tractable. I propose complexity metrics for multi-robot computation, in particular, and use a measure of algorithm accuracy that can be tied to the RSR to validate the claim from above. I test these metrics on a set of standard distributed algorithms from the literature for communication and navigation.

Finally, I present two new sets of distributed algorithms for boundary detection and dynamic task assignment. These algorithms have provable performance under ideal conditions, and robust performance under our RSR evaluation criteria.

Bibliography

[1] H. Abelson, R. Weiss, D. Allen, D. Coore, C. Hanson, G. Homsy, T. F. Knight Jr, R. Nagpal, E. Rauch, and G. J. Sussman. Amorphous computing. *Communications of the ACM*, 43:74–82, 2000.

[2] Norman Abramson. The aloha system: Another alternative for computer communications, April 1970.

[3] R. Arkin and T. Balch. *Line-of-Sight Constrained Exploration for Reactive Multiagent Robotic Teams*. 2002.

[4] Baruch Awerbuch. Complexity of network synchronization. *J. ACM*, 32:804–823, 1985.

[5] T. Balch and R. Arkin. Behavior-based formation control for multi-robot teams. *IEEE Transactions on Robotics and Automation*, 14:926–939, 1998.

[6] Julien Basch, Leonidas J. Guibas, and John Hershberger. Data structures for mobile data. pages 747–756, New Orleans, Louisiana, United States, 1997. Society for Industrial and Applied Mathematics.

[7] B. A. Bash and P. J. Desnoyers. Exact distributed voronoi cell computation in sensor networks. *Proceedings of the 6th international conference on Information processing in sensor networks*, pages 236–243, 2007.

[8] Maxim Batalin and Gaurav S. Sukhatme. Coverage, exploration and deployment by a mobile robot and communication network. pages 376–391, Palo Alto Research Center (PARC), Palo Alto, April 2003.

[9] Maxim Batalin and Gaurav S. Sukhatme. Coverage, exploration and deployment by a mobile robot and communication network. *Telecommunication Systems Journal, Special Issue on Wireless Sensor Networks*, 26:181–196, 2004.

[10] Maxim Batalin and Gaurav S. Sukhatme. The analysis of an effiient algorithm for robot coverage and exploration based on sensor network deployment. pages 3489–3496, Barcelona, Spain, April 2005.

[11] Maxim Batalin, Gaurav S. Sukhatme, and Myron Hattig. Mobile robot navigation using a sensor network. pages 636–642, New Orleans, Louisiana, April 2004.

[12] J. Bishop and E. Klavins. Collective sensing with self-organizing robots. In *Decision and Control, 2006 45th IEEE Conference on*, pages 4175–4181, 2006.

[13] E. Bonabeau, G. Theraulaz, and J. L. Deneubourg. Quantitative study of the fixed threshold model for the regulation of division of labour in insect societies. *Proceedings: Biological Sciences*, 263:1565–1569, 1996.

[14] Josh Broch, David A. Maltz, David B. Johnson, Yih-Chun Hu, and Jorjeta Jetcheva. A performance comparison of multi-hop wireless ad hoc network routing protocols. pages 85–97, Dallas, Texas, United States, 1998. ACM.

[15] R. Brooks. A robust layered control system for a mobile robot. *Robotics and Automation, IEEE Journal of [legacy, pre - 1988]*, 2:14–23, 1986.

[16] Y. U. Cao, A. S. Fukunaga, and A. Kahng. Cooperative mobile robotics: Antecedents and directions. *Autonomous Robots*, 4:7–27, 1997.

[17] S. Capkun, M. Hamdi, and J. P. Hubaux. Gps-free positioning in mobile ad hoc networks. *Cluster Computing*, 5:157–167, 2002.

[18] D. N. Coore. *Botanical Computing: A Developmental Approach to Generating Interconnect Topologies on an Amorphous Computer*. PhD thesis, Massachusetts Institute of Technology, 1994.

[19] Peter I. Corke, Stefan E. Hrabar, Ron Peterson, Daniela Rus, Srikanth Saripalli, and Gaurav S. Sukhatme. Autonomous deployment and repair of a sensor network using an unmanned aerial vehicle. pages 3602–3609, April 2004.

[20] Peter I. Corke, Stefan E. Hrabar, Ron Peterson, Daniela Rus, Srikanth Saripalli, and Gaurav S. Sukhatme. Deployment and connectivity repair of a sensor net. 2004.

[21] S. Corson and Anthony Ephremides. A distributed routing algorithm for mobile wireless networks. *Wireless Networks*, Volume 1:61–81, March 1995.

[22] S. Corson and J. Macker. Mobile ad hoc networking (manet): Routing protocol performance issues and evaluation considerations, 1999.

[23] J. Cortes, S. Martinez, T. Karatas, and F. Bullo. Coverage control for mobile sensing networks. In *Robotics and Automation, 2002. Proceedings. ICRA '02. IEEE International Conference on*, volume 2, pages 1327–1332, 2002.

[24] Karthik Dantu, Mohammad H. Rahimi, Hardik Shah, Sandeep Babel, Amit Dhariwal, and Gaurav S. Sukhatme. Robomote: Enabling mobility in sensor networks. pages 404–409. IEEE, April 2005.

[25] A.K. Das, R. Fierro, V. Kumar, J.P. Ostrowski, J. Spletzer, and C.J. Taylor. A vision-based formation control framework. *Robotics and Automation, IEEE Transactions on*, 18:813–825, 2002.

[26] J.P. Desai, V. Kumar, and J.P. Ostrowski. Control of changes in formation for a team of mobile robots. In *Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on*, volume 2, pages 1556–1561 vol.2, 1999.

[27] B. R. Donald. On information invariants in robotics. *Artificial Intelligence*, 72:217–304, 1995.

[28] G. Dudek, M. Jenkin, E. Milios, and D. Wilkes. A taxonomy for swarm robots. In *Intelligent Robots and Systems '93, IROS '93. Proceedings of the 1993 IEEE/RSJ International Conference on*, volume 1, pages 441–447 vol.1, 1993.

[29] H. Edelsbrunner, D. Kirkpatrick, and R. Seidel. On the shape of a set of points in the plane. *Information Theory, IEEE Transactions on*, 29:551–559, July 1983.

[30] Michael Erdmann. Towards task-level planning: Action-based sensor design, February 1992.

[31] D. Estrin, R. Govindan, J. Heidemann, and S. Kumar. Next century challenges: scalable coordination in sensor networks. *Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking*, pages 263–270, 1999.

[32] S. P. Fekete, M. Kaufmann, A. Kroeller, and K. Lehmann. A new approach for boundary recognition in geometric sensor networks. *Arxiv preprint cs.DS/0508006*, 2005.

[33] S. P. Fekete, A. Kroeller, D. Pfisterer, S. Fischer, and C. Buschmann. Neighborhood-based topology recognition in sensor networks. *Algorithmic Aspects of Wireless Sensor Networks: First International Workshop (ALGOSENSOR)*, page 123136, 2004.

[34] J.W. Fenwick, P.M. Newman, and J.J. Leonard. Cooperative concurrent mapping and localization. In *Robotics and Automation, 2002. Proceedings. ICRA '02. IEEE International Conference on*, volume 2, pages 1810–1817 vol.2, 2002.

[35] D. Fox, W. Burgard, H. Kruppa, and S. Thrun. A probabilistic approach to collaborative multi-robot localization. *Autonomous Robots*, 8:325–344, 2000.

[36] Jakob Fredslund and Maja J. Matari. Robot formations using only local sensing and control. Ban, Alberta, Canada, July 2001.

[37] B. P. Gerkey and M. J. Mataric. A formal analysis and taxonomy of task allocation in multi-robot systems. *The International Journal of Robotics Research*, 23:939, 2004.

[38] Brian P. Gerkey and Maja J. Mataric. Multi-robot task allocation: analyzing the complexity and optimality of key architectures. pages 3862–3868, 2003.

[39] Robert Ghrist, David Lipsky, Sameera Poduri, and Gaurav S. Sukhatme. Surrounding nodes in coordinate-free networks. 2006.

[40] D. Goldberg and M. Mataric. Interference as a tool for designing and evaluating multi-robot controllers. *AAAI/IAAI*, page 637642, 1997.

[41] D. M. Gordon. *Ants at Work: How an Insect Society is Organized.* WW Norton & Company, 2000.

[42] D. Gordon-Spears and W. Spears. Analysis of a phase transition in a physics-based multiagent system. *Lecture Notes in Computer Science*, page 193207, 2003.

[43] R. Grabowski and P. Khosla. Localization techniques for a team of small robots. In *Intelligent Robots and Systems, 2001. Proceedings. 2001 IEEE/RSJ International Conference on*, volume 2, pages 1067–1072 vol.2, 2001.

[44] Tian He, Chengdu Huang, Brian M. Blum, John A. Stankovic, and Tarek Abdelzaher. Range-free localization schemes for large scale sensor networks. pages 81–95, San Diego, CA, USA, 2003. ACM.

[45] B. Hlldobler and E. O. Wilson. *Journey to the Ants*. 1994.

[46] Andrew Howard, Maja J. Matari, and Gaurav S. Sukhatme. An incremental deployment algorithm for mobile robot teams. pages 2849–2854, Lausanne, Switzerland, October 2002.

[47] Andrew Howard, Maja J. Matari, and Gaurav S. Sukhatme. An incremental self-deployment algorithm for mobile sensor networks. *Autonomous Robots Special Issue on Intelligent Embedded Systems*, 13:113–126, 2002.

[48] Andrew Howard, Lynne E. Parker, and Gaurav S. Sukhatme. The sdr experience: Experiments with a large-scale heterogenous mobile robot team. Singapore, June 2004.

[49] Andrew Howard, Lynne E. Parker, and Gaurav S. Sukhatme. Experiments with large heterogeneous mobile robot team: Exploration, mapping, deployment and detection. *International Journal of Robotics Research*, 25:431–447, May 2006.

[50] C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed diffusion: A scalable and robust communication paradigm for sensor networks.

[51] T. Inukai. An efficient ss/tdma time slot assignment algorithm. *Communications, IEEE Transactions on [legacy, pre - 1988]*, 27:1449–1455, 1979.

[52] iRobot. Swarmbot. In *www.irobot.com*, 2002.

[53] B. Kannan and L. E. Parker. Fault-tolerance based metrics for evaluating system performance in multi-robot teams. *Proceedings of Performance Metrics for Intelligent Systems Workshop*, 2006.

[54] E. Klavins and E. Klavins. Programmable self-assembly. *Control Systems Magazine, IEEE*, 27:43–56, 2007.

[55] L. Kleinrock and J. Silvester. Optimum transmission radii for packet radio networks or why six is a magic number. *Proceedings of the IEEE National Telecommunications Conference*, 4:14.3, 1978.

[56] K. Kotay, D. Rus, M. Vona, and C. McGray. The self-reconfiguring robotic molecule. In *Robotics and Automation, 1998. Proceedings. 1998 IEEE International Conference on*, volume 1, pages 424–431 vol.1, 1998.

[57] M. J. Krieger, J. B. Billeter, and L. Keller. Ant-like task allocation and recruitment in cooperative robots. *Nature*, 406:992–5, 2000.

[58] A. Krller, S. P. Fekete, D. Pfisterer, and S. Fischer. Deterministic boundary recognition and topology extraction for large sensor networks. pages 1000–1009, 2006.

[59] F. Kuhn, R. Wattenhofer, and A. Zollinger. Ad hoc networks beyond unit disk graphs. *Wireless Networks*, pages 1–15.

[60] Fabian Kuhn, Thomas Moscibroda, and Roger Wattenhofer. Unit disk graph approximation. pages 17–23, Philadelphia, PA, USA, 2004. ACM.

[61] Fabian Kuhn, Roger Wattenhofer, Yan Zhang, and Aaron Zollinger. Geometric ad-hoc routing: of theory and practice. pages 63–72, Boston, Massachusetts, 2003. ACM.

[62] Newton Labs. Model 9000 vision system. In *www.newtonlabs.com/9000.htm*, 2001.

[63] Qun Li and Daniela Rus. Navigation protocols in sensor networks. *ACM Trans. Sen. Netw.*, 1:3–35, 2005.

[64] Xiang-Yang Li. Algorithmic, geometric and graphs issues in wireless networks. *Wireless communications and mobile computing(Print)*, 3:119–140, 2003.

[65] Xiang-Yang Li, G. Calinescu, and Peng-Jun Wan. Distributed construction of a planar spanner and routing for ad hoc wireless networks. In *INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 3, pages 1268–1277 vol.3, 2002.

[66] Xiang-Yang Li, Peng-Jun Wan, and Yu Wang. Power efficient and sparse spanner for wireless ad hoc networks. In *Computer Communications and Networks, 2001. Proceedings. Tenth International Conference on*, pages 564–567, 2001.

[67] Nancy A. Lynch. *Distributed Algorithms*. Morgan Kaufmann Publishers Inc., 1996.

[68] S. Madden, J. Hellerstein, and W. Hong. Tinydb: In-network query processing in tinyos. *Intel Research, IRB-TR-02-014, October*, 2002.

[69] S. R. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. Tinydb: an acquisitional query processing system for sensor networks. *ACM Transactions on Database Systems (TODS)*, 30:122–173, 2005.

[70] M. V. Marathe, H. Breu, H. B. Hunt III, S. S. Ravi, and D. J. Rosenkrantz. Simple heuristics for unit disk graphs. *Networks*, 25:5968, 1995.

[71] J. McLurkin and J. Smith. Distributed algorithms for dispersion in indoor environments using a swarm of autonomous mobile robots. *7th International Symposium on Distributed Autonomous Robotic Systems (DARS)*, 2004.

[72] J. McLurkin and D. Yamins. Dynamic task assignment in robot swarms. *Proceedings of Robotics: Science and Systems, June*, 8, 2005.

[73] James McLurkin. *Stupid Robot Tricks: A Behavior-Based Distributed Algorithm Library for Programming Swarms of Robots*. PhD thesis, Massachusetts Institute of Technology, 2004.

[74] James McLurkin, Jennifer Smith, James Frankel, David Sotkowitz, David Blau, and Brian Schmidt. Speaking swarmish: Human-robot interface design for large swarms of autonomous mobile robots. In *AAAI Spring Symposium*, March 2006.

[75] D. P. Mehta and S. Sahni. *Handbook of Data Structures and Applications*. Chapman & Hall/CRC, 2005.

[76] David Moore, John Leonard, Daniela Rus, and Seth Teller. Robust distributed network localization with noisy range measurements. pages 50–61, Baltimore, MD, USA, 2004. ACM.

[77] John Nagle. Congestion control in ip/tcp internetworks. *SIGCOMM Comput. Commun. Rev.*, 14:11–17, 1984.

[78] R. Nagpal. *Programmable Self-Assembly: Constructing Global Shape using Biologically-inspired Local Interactions and Origami Mathematics*. Ph.d. thesis, Massachusetts Institute of Technology, 2001.

[79] R. Nagpal, H. Shrobe, and J. Bachrach. Organizing a global coordinate system from local information on an ad hoc sensor network. *Proc. of Information Processing in Sensor Networks (IPSN)*, 2003.

[80] Delphine Nain, Noshirwan Petigara, and Hari Balakrishnan. Integrated routing and storage for messaging applications in mobile ad hoc networks. *Mob. Netw. Appl.*, 9:595–604, 2004.

[81] S. Nath, P. B. Gibbons, S. Seshan, and Z. Anderson. Synopsis diffusion for robust aggregation in sensor networks. 2008.

[82] S. Nouyan. Agent-based approach to dynamic task allocation. *Ant Algorithms: Third International Workshop, ANTS*, 2463:221–226, 2002.

[83] J. M. OKane and S. M. LaValle. Dominance and equivalence for sensor-based agents. *Urbana*, 51:61821.

[84] J. M. OKane and S. M. LaValle. On comparing the power of mobile robots. *Proceedings of Robotics: Science and Systems*, 2006.

[85] J. M. O'Kane and S. M. LaValle. Comparing the power of robots. *The International Journal of Robotics Research*, 27:5, 2008.

[86] R. Olfati-Saber, J.A. Fax, and R.M. Murray. Consensus and cooperation in networked multi-agent systems. *Proceedings of the IEEE*, 95:215–233, 2007.

[87] Joseph O'Rourke. *Computational geometry in C (2nd ed.)*. Cambridge University Press, 1998.

[88] Esben Ostergaard, Gaurav S. Sukhatme, and Maja J. Matari. Emergent bucket brigading - a simple mechanism for improving performance in multi-robot constrained-space foraging tasks. Montreal, Canada, May 2001.

[89] L. E. Parker. Current state of the art in distributed autonomous mobile robotics. *Distributed Autonomous Robotic Systems*, 49:312, 2000.

[90] D. Payton, R. Estkowski, and M. Howard. Compound behaviors in pheromone robotics. *Robotics and Autonomous Systems*, 44:229–240, 2003.

[91] C.E. Perkins and E.M. Royer. Ad-hoc on-demand distance vector routing. In *Mobile Computing Systems and Applications, 1999. Proceedings. WMCSA '99. Second IEEE Workshop on*, pages 90–100, 1999.

[92] P. Pirjanian, C. Leger, E. Mumm, B. Kennedy, M. Garrett, H. Aghazarian, S. Farritor, and P. Schenker. Distributed control for a modular, reconfigurable cliff robot. In *Robotics and Automation, 2002. Proceedings. ICRA '02. IEEE International Conference on*, volume 4, pages 4083–4088 vol.4, 2002.

[93] Sameera Poduri, Sundeep Pattem, Bhaskar Krishnamachari, and Gaurav S. Sukhatme. A unifying framework for tunable topology control in sensor networks, 2005.

[94] Sameera Poduri and Gaurav S. Sukhatme. Achieving connectivity through coalescence in mobile robot networks. October 2007.

[95] Sameera Poduri and Gaurav S. Sukhatme. Latency analysis of coalescence in robot groups. pages 3295–3300, 2007.

[96] Franco P. Preparata and Michael I. Shamos. *Computational geometry: an introduction.* Springer-Verlag New York, Inc., 1985.

[97] Nissanka B. Priyantha, Anit Chakraborty, and Hari Balakrishnan. The cricket location-support system. pages 32–43, Boston, Massachusetts, United States, 2000. ACM.

[98] S. Rajsbaum and J. Urrutia. Some problems in distributed computational geometry. *6th International Colloquium on Structural Information and Communication Complexity (SIROCCO), Lacanau, France, July*, page 233248, 1999.

[99] Craig W. Reynolds. Flocks, herds and schools: A distributed behavioral model. *SIGGRAPH Comput. Graph.*, 21:25–34, 1987.

[100] S.I. Roumeliotis and G.A. Bekey. Distributed multirobot localization. *Robotics and Automation, IEEE Transactions on*, 18:781–795, 2002.

[101] D. Rus, B. Donald, and J. Jennings. Moving furniture with teams of autonomous robots. *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 235–242, 1995.

[102] D. Rus and M. Vona. A physical implementation of the self-reconfiguring crystalline robot. In *Robotics and Automation, 2000. Proceedings. ICRA '00. IEEE International Conference on*, volume 2, pages 1726–1733 vol.2, 2000.

[103] Daniela Rus, Zack Butler, Keith Kotay, and Marsette Vona. Self-reconfiguring robots. *Commun. ACM*, 45:39–45, 2002.

[104] M. Schwager, J.-J. Slotine, and D. Rus. Decentralized, adaptive control for coverage with networked robots. In *Robotics and Automation, 2007 IEEE International Conference on*, pages 3289–3294, 2007.

[105] Peng Song and V. Kumar. A potential field based approach to multi-robot manipulation. In *Robotics and Automation, 2002. Proceedings. ICRA '02. IEEE International Conference on*, volume 2, pages 1217–1222, 2002.

[106] W. M. Spears, D. F. Spears, J. C. Hamann, and R. Heil. Distributed, physics-based control of swarms of vehicles. *Autonomous Robots*, 17:137–162, 2004.

[107] W. M. Spears, D. F. Spears, R. Heil, W. Kerr, and S. Hettiarachchi. An overview of physicomimetics. *Lecture Notes in Computer Science-State of the Art Series*, 3342, 2005.

[108] W.M. Spears, W.M. Spears, R. Heil, R. Heil, D.F. Spears, and D. Zarzhitsky. Physicomimetics for mobile robot formations. In *Autonomous Agents and Multiagent Systems, 2004. AAMAS 2004. Proceedings of the Third International Joint Conference on*, pages 1528–1529, 2004.

[109] J. Spletzer, A.K. Das, R. Fierro, C.J. Taylor, V. Kumar, and J.P. Ostrowski. Cooperative localization and control for multi-robot manipulation. In *Intelligent Robots and Systems, 2001. Proceedings. 2001 IEEE/RSJ International Conference on*, volume 2, pages 631–636 vol.2, 2001.

[110] S. Thrun, W. Burgard, and D. Fox. A probabilistic approach to concurrent mapping and localization for mobile robots. *Autonomous Robots*, 5:253–271, 1998.

[111] J. Tsitsiklis, J. Tsitsiklis, D. Bertsekas, D. Bertsekas, and M. Athans. Distributed asynchronous deterministic and stochastic gradient optimization algorithms. *Automatic Control, IEEE Transactions on*, 31:803–812, 1986.

[112] Grey Walter. An imitation of life. *Scientific American*, 182(5):42–45, May 1950.

[113] Y. Wang and X. Y. Li. Localized construction of bounded degree and planar spanner for wireless ad hoc networks. *Mobile Networks and Applications*, 11:161–175, 2006.

[114] Yu Wang and Xiang-Yang Li. Distributed spanner with bounded degree for wireless ad hoc networks. In *Parallel and Distributed Processing Symposium., Proceedings International, IPDPS 2002, Abstracts and CD-ROM*, pages 194–201, 2002.

[115] R. Wattenhofer. Ad-hoc and sensor networks: Worst-case vs. average-case. *Proceedings of International Zurich Seminar on Communications*, page 156159, 2004.

[116] Jens Wawerla, Gaurav S. Sukhatme, and Maja J. Matari. Collective construction with multiple robots. pages 2696 – 2701, Lausanne, Switzerland, October 2002.

[117] Alec Woo, Terence Tong, and David Culler. Taming the underlying challenges of reliable multihop routing in sensor networks. pages 14–27, Los Angeles, California, USA, 2003. ACM.

[118] M. Yim, D.G. Duff, and K.D. Roufas. Polybot: a modular reconfigurable robot. In *Robotics and Automation, 2000. Proceedings. ICRA '00. IEEE International Conference on*, volume 1, pages 514–520 vol.1, 2000.

[119] M. Yim, Wei-Min Shen, B. Salemi, D. Rus, M. Moll, H. Lipson, E. Klavins, and G.S. Chirikjian. Modular self-reconfigurable robot systems [grand challenges of robotics]. *Robotics & Automation Magazine, IEEE*, 14:43–52, 2007.