Distributed Area Search with a Team of Robots

by

Velin K. Tzanov

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

Bachelor of Science in Computer Science and Engineering

and

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2006

Author . . . .
Department of Electrical Engineering and Computer Science
May 26, 2006

Certified by . . . .
Jonathan R. Bachrach
Research Scientist
Thesis Supervisor

Accepted by . . . .
Arthur C. Smith
Chairman, Department Committee on Graduate Theses

Distributed Area Search with a Team of Robots

by

Velin K. Tzanov

Submitted to the Department of Electrical Engineering and Computer Science
on May 26, 2006, in partial fulfillment of the
requirements for the degrees of
Bachelor of Science in Computer Science and Engineering
and
Master of Engineering in Electrical Engineering and Computer Science

Abstract

The main goal of this thesis is to demonstrate the applicability of the distributed systems paradigm to robotic systems. This goal is accomplished by presenting two solutions to the Distributed Area Search problem: organizing a team of robots to collaborate in the task of searching through an area. The first solution is designed for unreliable robots equipped with a reliable GPS-style localization system. This solution demonstrates the efficiency and fault-tolerance of this type of distributed robotic systems, as well as their applicability to the real world. We present a theoretically near-optimal algorithm for solving Distributed Area Search under this setting, and we also present an implementation of our algorithm on an actual system, consisting of twelve robots. The second solution is designed for a completely autonomous system, without the aid of any centralized subsystem. It demonstrates how a distributed robotic system can solve a problem that is practically unsolvable for a single-robot system.

Thesis Supervisor: Jonathan R. Bachrach
Title: Research Scientist

Acknowledgments

This thesis would not be possible without the invaluable help of many people. First and foremost, I would like to thank my thesis supervisor Dr. Jonathan Bachrach for guiding my research and supporting me in every possible way.

I would also like to thank Rangel Dokov for hand-assembling most of the robots and for spending days, nights and long weekends working with me on the theoretical algorithms and the hardware system.

I am also extremely grateful to my wife, Gergana, not only for being beside me during all these years, but also for her numerous suggestions for improvements both on my proofs and on my writing. Without her help, this thesis would have been of much lower quality.

Finally, I would like to thank all my friends and relatives and especially the students in Dr. Bachrach's research group for providing constructive feedback on earlier versions of my work.

Contents

1 Introduction **13**

1.1 Problem Definition . . . 14

1.2 Two Different Models . . . 16

1.3 Organization . . . 18

2 Related Work **19**

2.1 Overview of Distributed Systems and Robotics . . . 19

2.2 Collaborative Robotics Research . . . 20

2.2.1 Distributed Area Coverage . . . 21

2.2.2 Distributed Area Mapping . . . 23

3 Location-Aware System **27**

3.1 Model Assumptions . . . 28

3.2 Environment Representation . . . 32

3.3 Goals and Lower Bounds . . . 34

3.4 Communication Protocol . . . 39

3.4.1 Data Structures . . . 40

3.4.2 Messages . . . 41

3.5 Compact Coverage Algorithm . . . 43

3.5.1 Formal Definition . . . 49

3.5.2 Proof of Correctness and Performance Bounds . . . 52

3.6 Simulation Comparison . . . 59

4 Hardware Implementation **65**

4.1 Mark III Robots . . . 66

4.2 Crickets . . . 68

4.3 Localization System . . . 69

4.4 System Integration . . . 72

4.5 Results and Evaluation . . . 74

5 Autonomous System **77**

5.1 Model Assumptions . . . 79

5.2 Environment Representation . . . 81

5.3 DFS Exploration Algorithm . . . 84

5.4 Proof of Correctness . . . 89

5.5 Complexity Bound . . . 90

5.6 Hardware Implementation . . . 93

5.7 Discussion of Reliability . . . 97

5.7.1 Failure Detection . . . 97

5.7.2 Consistency of Sensor Readings . . . 99

6 Future Work **101**

6.1 Location-Aware System . . . 101

6.2 Autonomous System . . . 105

7 Conclusion **107**

List of Figures

3-1 A case suggesting an $\Omega(B)$ lower bound. 38

3-2 An illustration of the meaning of “clockwise” direction on the front line. 45

3-3 Example run of the CCA. Screenshot 1 45

3-4 Example run of the CCA. Screenshot 2 46

3-5 Example run of the CCA. Screenshot 3 46

3-6 Example run of the CCA. Screenshot 4 47

3-7 Example run of the CCA. Screenshot 5 47

3-8 Example run of the CCA. Screenshot 6 48

3-9 Example run of the CCA. Screenshot 7 48

3-10 Test case 1: Floor plan in the Stata Center. 60

3-11 Test case 2: Part of the Caribbean Sea. 61

3-12 Test case 3: A recreational park plan. 61

3-13 Test case 4: A living room. 62

4-1 An individual off-the-shelf Mark III robot. 66

4-2 An individual off-the-shelf Cricket. 69

4-3 A coupled pair of a Cricket and a Mark III robot. 72

4-4 A test maze with 6 robots. 74

4-5 Part of a test maze with 9 robots.. 75

5-1 An illustration of the different types of C sides. 82

5-2 The order in which the DFS Exploration Algorithm covers triangles in a corridor network. 85

5-3 The order in which the DFS Exploration Algorithm covers triangles in a more wide-open case. 86
5-4 A modified Cricket. 94
5-5 A modified Mark III robot. 95
5-6 A coupled pair of a modified Cricket and a modified Mark III robot. . 96

List of Tables

3.1 Simulation results. 63

4.1 Pattern of discoveries in the real-world prototype system. 76

Chapter 1

Introduction

Distributed systems are a large area of research in Computer Science. This is mostly due to the fact that distributed systems provide two very desirable properties, which we cannot achieve effectively by other means. The first property is load-balancing and time efficiency: if the components of the distributed system share the workload they would be able to accomplish the whole task in less time. The second property is fault-tolerance: if one or more of the individual components fail, the whole system would be able to continue to function correctly. In a sense, distributed systems provide us means by which we can build reliable systems out of unreliable components: a paradigm with enormous applicability.

Traditionally, research in distributed systems has been focused on computer systems and especially on software systems. This branch of the area is more than 20 years old and this was enabled by the wide availability of hardware since the late 1970s, early 1980s. A relatively new, emerging branch of distributed systems is sensor networks and monitoring applications. This branch's development was also enabled by the relatively recent hardware availability. Five years ago, when cheap sensors nodes were not available, there was practically no research in this area at all.

This thesis focuses on applying the distributed systems paradigm to a third set of systems: robotic systems. Unlike in the two branches above, there has been limited amount of research in distributed (or sometimes called collaborative) robotics. The advantages of using a team of robots have been mentioned numerous times [9, 15, 22], and recently there have been several publications exploring theoretical models for large teams of robots [7, 11, 14, 25]. However, until now we know of only one real system with double-digit number of actual, real-world robots [16]. This, we believe, is also strongly related to the lack of hardware availability. Autonomous robotic systems are extremely rare and it is considered an exceptional success even when a single-robot system performs well enough to be commercialized. We claim that the latter is starting to change [21] and that we have all the prerequisites to start doing more research in distributed robotic systems now.

1.1 Problem Definition

This thesis demonstrates the various applications of distributed systems to robotics by presenting two solutions to the Distributed Area Search problem: organizing a team of robots to collaborate in the task of searching through an area. Searching, in this case, is defined as looking for something specific throughout the whole area: it might be an object, a chemical agent, a given type of terrain, etc. This searching is accomplished through a specific scanning primitive, which checks whether the object is within some distance r of the present position of the robot. In order to be as general as possible, we abstract away this primitive and focus on the rest of the operation, which is making sure that the robots have physically traveled within some distance of every possible location in the area.

Area Search is a superset of the Area Coverage problem and is a subset of the Area Mapping problem. Area Coverage is what a robotic vacuum cleaner does, which is an instance of Area Search with a scanning primitive distance of zero. Area Mapping means building a map of the area, which inevitably includes Area Search, as the robot has to visit (within some distance) every location on the area in order to map it. In addition, Area Mapping requires that robots be aware of their global coordinates when they inspect new regions: a requirement that Area Search does not impose. The latter fact is crucial to the algorithm described in chapter 5 below.

The Distributed Area Search problem was chosen for a number of reasons. First, it is an actual real-world problem that covers a number of possible applications. Second, it is clearly a problem that allows for improvement through parallelism, but it is not obvious how this improvement can be achieved, nor how substantial the improvement can be. Another important property of Area Search is that it has a clear definition and a clear set of metrics. We can accurately measure the percentage of area covered and the time it takes to accomplish the task. Having clear, quantitative ways to measure success is very important for a project of this sort. Another factor contributing to our choice of Area Search was the problem's relative simplicity. The task of the individual robots is not overly complicated and so we were able to focus on the distributed aspects of the project and did not have to spend too much time getting even the single-robot system to work.

Last, but not least, there are two important factors that provided a good basis for our project. First, there was research in similar areas already: there are numerous papers dealing with collaborative Area Coverage and collaborative Area Mapping. Unfortunately, they rarely build actual hardware implementation of their systems and when they do, they tend to use expensive robots and end up having no more than three of them in the actual system [22, 27]. Second, our project had low-cost hardware requirements. Nowadays, there are even commercially successful, low-cost robots that do Area Coverage: more than a million copies of the Roomba robotic vacuum cleaner have been sold until now, and the cost of one such robot is as little as $150 [21]. However, such low-cost robots have never been used in a massively distributed setting so far, to the best of our knowledge. In this thesis we bridge this gap, first by advancing the research in collaborative robotics, moving it closer to research in distributed systems, and second by building an actual Area Search system consisting of 12 low-cost robots. In a completely separate system (described in chapter 5 of this document) we also explore how the availability of a larger number of robots can enable us to approach some problems in ways impossible for smaller-scale systems.

1.2 Two Different Models

The main challenge in building any robotic Area Search system is the well-known problem of robot localization, i.e. the ability of a robot to infer its location, at least compared to its previous locations (if not globally), without having a map initially. In addition, the distributed nature of our robotic platform amplifies this problem to a multi-robot localization and map-synchronization problem, i.e. even if each robot knows where it is on its own locally-built map, the different robots in the system should be able to match their maps, build a common map and infer each other's locations.

How one approaches the localization problem is fundamental to what the whole Area Search system will look like and what the focus of that system is going to be. This thesis considers two different options for what localization infrastructure is available to the robot team. These two options define two very different approaches to addressing the localization problem, which in turn become the bases of two completely different systems. These two systems focus on different aspects and benefits of the distributed systems class, and thus, they allow us to come up with two very different (but complementary, not contradicting) conclusions regarding the applicability of distributed systems to robotics.

In the first case, we assume the existence of a global localization system (similar to the GPS), which can provide robots with their coordinates on demand. The system is assumed to work at any location and at any time, regardless of the existence or liveness of the other robots.[1] The presence of this universal, centralized localization system directly solves the single- and multi-robot localization problems and it also reduces map-synchronization to the much simpler problem of data sharing (i.e., now the robots only have to tell one another what they know about the single, global map, instead of also trying to align their coordinates, orientations and scales).[2] The convenient localization system also allows us to move the problem from a continuous world, in which probabilities and magnitudes of measurement errors play an important

[1]See chapter 3 for more details on the exact assumed properties of the localization system.

[2]See section 3.4 for a detailed discussion on the protocol used for data sharing.

role, to a much more discrete world, in which we can use a discrete algorithm[3] similar to a state machine to solve the problem.

Solving Area Search in such location-awareness conditions with a single robot is relatively easy. That is why, under these assumptions, we focus on the efficiency and fault-tolerance properties of our Distributed System. The measure of success of this location-aware system is what improvements in performance we observe (measured by time taken to accomplish the task), both with a fully functional system and in cases where robots fail.

The assumption of the existence of a reliable global localization system is realistic for many applications (such as large-scale, open-field landmine removal, where GPS can be used), but clearly this is not the case in many possible scenarios (such as exploration of underground mineral mines, as in [23]). In order to address these cases we consider a second model to base our Distributed System on and that is a model with no centralized subsystem of any sort, where the system is completely autonomous and where the robots share no common pre-existing infrastructure of any kind.

In this model, robots are allowed to interact with each another (e.g. measure the pairwise distances between them), but still, determining their global coordinates is prone to accumulating error. Moreover, we claim that under this model, without any infrastructure (such as recognizable stationary landmarks), it is practically impossible to solve Area Search with a single robot, or even with a small number of robots. That is why the focus of this second Distributed System described in this thesis is only on the correctness of the solution, not on its efficiency or fault-tolerance. The significant result of this second part of the thesis is that distributed systems allow us to solve problems, which are otherwise unsolvable.

In addition to the above two theoretical models, this work includes an experimental part, which consists of a complete hardware implementation of a system of the first model and a proof-of-concept mini-system demonstrating the feasibility of systems based on the second model.

[3]See section 3.5.

1.3 Organization

This thesis is organized as follows. Chapter 2 describes the related work in the field, as well as the papers whose ideas are applied in this thesis. Chapter 3 describes the theoretical basis of our first system, the one based on the location-aware model, in detail. It includes sections on the details of the model and the communication protocol, as well as description, bounds and proofs of the algorithm making the decisions where each robot should go in every situation. The chapter concludes with test evaluation of the algorithm in a software simulated environment. Chapter 4 describes the hardware implementation of the location-aware system. It elaborates on the specifics of the robots and the other devices we used and it also discusses to what extent the hardware reality matched the assumptions of the model.

Chapter 5 is focused on the second system, built under the autonomous system model, and is similar in structure to chapter 3, including sections on the details of the model, the specifics of the algorithm and a feasibility evaluation of a hardware implementation. Chapters 6 and 7 conclude the thesis by exploring possibilities for extending or applying the work described here, as well as drawing conclusions based on the accomplished theoretical and experimental work.

Chapter 2

Related Work

This chapter places the work presented in this thesis in its historical context. Section 2.1 makes a high-level overview of and comparison between the two research areas related to this thesis: distributed systems and robotics. Section 2.2 lists and analyzes a number of particular pieces of research related to our work.

2.1 Overview of Distributed Systems and Robotics

The fields of distributed systems and robotics are more than twenty[1] years old. Each one of them has matured in its own way and is now an established field in Computer Science with many respected annual conferences and journals and a whole research community. However, the two fields have matured in two different directions.

Distributed systems are traditionally run on relatively reliable, off-the-shelf computer hardware. The capabilities and limitations of this hardware are mostly well-studied and well-known. Thus, research in distributed systems rarely focuses on theoretical arguments related to these capabilities and limitations. Instead, the field is much more focused on scalability and performance improvement. Naturally, scalability concerns bring the goals of efficiency and fault-tolerance: the properties distributed systems are most well-known for. That is why distributed systems research can, in general, be classified as very quantitative in nature, dealing mostly with precise

[1]Arguably more than thirty.

numbers and/or asymptotic factors.

In the field of robotics the situation is different. There is a huge variety of hardware and the capabilities and limitations of this hardware are still mostly unclear. Robotics research is constantly expanding the realm of what can be done with robots, and there is a general lack of platform standardization. Thus, the dependability of the hardware is often questionable, and a lot of research work in robotics is concerned with how to account for (and react to) possible misbehaviors of the hardware platform. Given that even single-robot systems have a lot of issues to deal with, multi-robot systems have historically been very rare. That is why scalability and asymptotic behavior have rarely (if ever) been goals of robotics research. Quantitative "head-to-head" comparisons between systems, as are common in distributed systems, are extremely rare in robotics. That is why this field can, in general, be classified as mostly qualitative in nature.

This thesis is an attempt at creating an inter-disciplinary piece of research, belonging at the same time to robotics and to distributed systems. Given the wide gap between the styles of the two fields, this is not an easy task. Fortunately, the fundamental reason for this divide, which is the lack of relatively reliable, off-the-shelf hardware, is rapidly beginning to disappear. The Roomba robotic vacuum cleaner is already a commonplace and it costs as low as $150 [21]. We take this as a sure signal that the field of distributed robotic systems is going to evolve rapidly in the next few years.

2.2 Collaborative Robotics Research

Before proceeding with listing specific papers, we should mention that our research builds upon years of accumulated wisdom both by the distributed systems community and by the robotics community. Many components of our systems are following classical designs[2] coming from these two fields.

[2]For example, things taught in undergraduate and/or graduate classes, without any references to research papers.

One example is our data sharing protocol, described in section 3.4. It is based on a commonplace idea in distributed systems research, which tells us the optimal way of sharing data, when the agents in the distributed system accumulate that data in a monotonically growing fashion. Each piece of data is given a unique, monotonically increasing version number. Then the agents communicate only the latest version numbers they have. Whenever an agent A learns that it possesses more data than some other agent B (because A has a higher version number), A simply sends the difference in the data sets to B. By repeatedly broadcasting their version numbers, the agents make sure that every agent is up to date all of the time, as long as the whole system is connected (in terms of communication links). In cases in which not everyone can communicate with everyone else, the system effectively distributes the date in a "gossiping" manner.

Another example of following a classical design in our systems are the simple differential feedback controllers we employ in the navigation system of our location-aware robots in chapter 4 and the wall-following mechanism of our autonomous robots in section 5.6.

Besides the commonplace techniques mentioned above, our work also relates to or builds upon the published work of many others. Section 2.2.1 below discusses the publications related to our work on the location-aware system described in chapters 3 and 4. Section 2.2.2 does the same for our work on the autonomous system described in chapter 5.

2.2.1 Distributed Area Coverage

The benefits of using a collaborative team of robots have been mentioned numerous times for over a decade now. As early as 1995 Mataric et al. presented a system in which a pair of robots collaborates in the task of pushing a box [15]. Only a year later Guzzoni et al. presented a multi-robot system capable of navigating office environments, shortening the run time of the system by having the robots collaborate in their task [9]. These papers, as well as the many similar ones following them, provided a basis for thinking about collaborative multi-robot systems. Nevertheless, these systems were still very far from the field of distributed systems, as they did not address scalability at all.[3] This pattern of qualitative, but not quantitative papers, presenting designs of simple multi-robot systems without addressing scalability, seems to be prevalent in the nineties.

During the last four years, more and more robotics research publications got closer to discussing real distributed robotic systems. In fact, during that period a number of papers addressing problems very specific to distributed robotic systems appeared. These include the works of Lerman et al. on dynamic task allocation [11], Ulam et al. and Martinson et al. on division of labor [25, 14] and Gerkey and Mataric on multi-robot coordination [7]. The common pattern among these works is the consideration of scalability with more robots as an important property of the system. This is most often expressed by the inclusion of simulation runs featuring larger sets of robots than physically available.

On the less theoretical and more practical side, there has been substantial progress towards real distributed robotic systems too. In his Master's thesis James McLurkin presents a number of behaviors for a swarm of robots [16]. The most intriguing part of his work is the real-world implementation of an actual swarm of more than 100 robots. Nevertheless, McLurkin's thesis focuses on emerging behaviors, not on classical systems metrics, such as efficiency. Thus, his work does not bridge the fields of robotics and distributed systems in the way that we are trying to do. His focus is still on the qualitative side of what can and what cannot be done, not on the quantitative side of how well exactly it can be done.

Of course, there have been several papers addressing problems similar to Distributed Area Search, most often Distributed (or Collaborative) Area Coverage. Min and Yin present a greedy solution with excellent fault-tolerance [17]. The authors claim their algorithm is also efficient, but there is no theoretical analysis supporting this claim. Min and Yin work with a discrete grid world representation, an idea which we apply in our system as well. They also present a software simulation of their algorithm, but unfortunately it is very small-scale, so it is hard to draw any reliable

[3]This is completely understandable, given the low hardware availability at the time.

conclusions on the actual efficiency of their system.

Luo and Young also address Distributed Area Coverage on a grid representation [12]. Their setup is similar to the one of Min and Yin (and to ours as well), but their planning algorithm is different. Luo and Young proposed a solution based on neural networks, which is very intriguing, but nevertheless lacking any theoretical support for why it is efficient. Luo and Young also simulate their system in software, but the small scale and lack of any real-world hardware prevent us from reaching a positive conclusion as to the applicability and scalability of their proposed system.

Most recently, Rekleitis et al. present a system for solving Distributed Area Search based on Boustrophedon decomposition [20]. Their model, however, is different from ours in that they assume communication connectivity based on the presence of uninterrupted direct line of sight. This limits their flexibility and thus they end up with an algorithm in which the robots have additional incentives to stick together, beyond what might be optimal. In addition, there are other problematic issues. They have a pair of specialized robots in the system (called explorers), which explore the boundary of the area. It is unclear what happens if one of these robots fails, so the fault-tolerance level of their system is questionable. Also, just like the other works on this problem, they do not have a theoretical proof of the efficiency of their system, nor do they have a real-world implementations to test the feasibility of their assumptions.

In the following chapters 3 and 4 we present a fault-tolerant Distributed Area Search system that, unlike all systems outlined above, has a theoretical proof of its efficiency with a tight asymptotic upper bound. The scalability of the system is demonstrated by large-scale simulation and the realism of its assumptions is demonstrated by experiments on a real-world system with double-digit number of robots. This, in our opinion, brings our robotics research much closer to the quantitative ways of distributed systems research.

2.2.2 Distributed Area Mapping

Our second system attempts to solve Distributed Area Search under the completely autonomous model (i.e., no external localization system). Under this model the problem is much harder[4], which is why we do not attempt to build an efficient and highly fault-tolerant solution, but only to construct a solution, which is guaranteed to work (under a set of assumptions). The closest problem to ours, addressed in recent research papers, is the problem of Distributed Area Mapping under the same autonomous model (sometimes referred to as Autonomous Mapping and Exploration). The works of Yamauchi [27] and Simmons et al. [22], while making very important theoretical contributions, never build a real system with more than three robots.

The root cause of this phenomenon, particularly in Area Mapping, is that because the problem itself is hard, most people attempt to solve it by using expensive, sensor-rich robots [1, 5, 19, 28], which drives the cost of a single robot up and thus disallows a cost-effective distributed solution.

Our approach to this problem[5] is radically different. We design our system for robots with minimal sensory equipment, which would allow us to build a more distributed (in terms of number of robots) system. Then, having a highly distributed system, we can solve the hard problems of localization and loop closure in a way that would be impossible under a system with very limited number of robots.

Our algorithm for our autonomous system builds upon the work of Rekleitis et al. [19] by using the analysis presented by Kurazume et al. [10]. The paper by Rekleitis et al. presents a two-robot system for solving Area Search. Their approach is to use large, expensive robots with good vision equipment, which can literally "see" one another all the time. Using these two robots, Rekleitis et al. build a triangulation of the area, step-by-step, one triangle at a time, by using line-of-sight detection between the robots.[6] This allows the robots to cover the area without overlapping adjacent triangles in the triangulation. Nevertheless, their system has numerous problems related to the inability of the robots to detect loop closures or to backtrack effectively.

We borrow the idea of building a triangulation of the area, but instead of applying it to a pair of expensive robots, we propose applying it to a swarm of cheap robots with

[4]See chapter 5 for a detailed discussion.

[5]In fact, to a slightly easier problem, as Distributed Area Search under the autonomous model is a subset of Distributed Area Mapping.

[6]See section 5.2 for a detailed explanation.

minimal sensory equipment. The large number of robots would allow some of them to remain at various checkpoints in the area, which would enable reliable loop closure detection. This, we claim, would allow building an actual autonomous Distributed Area Search system with guaranteed correctness (under a set of assumptions).

The work of Kurazume et al. complements the above-mentioned contribution by providing us with a means of analysis of the local interactions between the robots in our system. In their paper, Kurazume et al. analyze the ability of robots to serve as temporary stationary nodes during migration of their whole 3-member group of robots.

An important result of theirs, which we use heavily in our system, is that even though the global coordinates of the moving group drift away unboundedly as time progresses, the local, relative errors of the robot movements are within tight bounds. This result is especially important in our system, which solves Area Search without solving Area Mapping. We do not need a guarantee that the global coordinates of the robots are accurate, as long as we know that the relative coordinates of adjacent robots are known within small errors. This would guarantee us that we have no coverage "holes" between adjacent robots, which would be enough for our purposes, and this is exactly what the results by Kurazume et al. provide us.

Chapter 3

Location-Aware System

This chapter describes a design of a system for solving the Distributed Area Search problem under the assumption of the presence of a reliable, universal localization system. The system is designed for a team of several (between 2 and around 30) low-cost, unreliable robots, which are able to function independently, but which also have the ability to communicate with one another and to localize themselves on a global coordinate system. Each one of these robots can fail at any point in time, as can several of the robots simultaneously. However, the robots are assumed stable enough that one can reasonably expect most of them to complete the assignment without failing. The only strict requirement on survivability, however, is simply that at least one robot survives until the completion of the task.

It is important to note that the system is not designed for expensive robots, equipped with quality sensors of the environment, as our algorithm makes no use of sensory information about the environment outside the present position of the robot. That is, if you have robots with visual, ultra-sound, infra-red, laser and/or other sensors for detecting features of the surrounding world, and you would like them to do Area Search efficiently, then our system design would not be the best choice for you. Sensory data gives you a lot of information, which our algorithm simply does not take into account. For example, just by scanning the environment with advanced sensors you might be able to build a map of the area without visiting all those regions on your map. This would likely allow you to develop algorithms, which knowing the map in advance would perform much better than our algorithm, which uses no prior map knowledge.

The system also is not designed for large swarms of virtually sensorless robots. In this case, some variation of a random walk is likely to do an excellent job. On the other hand, a planning algorithm, such as the one described in this chapter, would face enormous communication strain in such a dense population of robots that the complexity of handling all the information would likely be more costly than the savings achieved through planning.

The rationale behind this choice is our estimate that a system of the chosen scale and capabilities has the optimal cost/benefit ratio. The real-world prototype of our system features 12 robots capable of communicating with one another and localizing themselves on the global coordinate system. The average cost of a robot plus its additional sensory devices is below \$300. This gives a total cost of the system below \$4000.

With the same budget one can afford to buy no more than two or three expensive robots with advanced sensors, which are very unlikely to match the performance of a team of 12 robots running an implementation of our algorithms. Fault-tolerance would also be a bigger problem for a system built of fewer robots.

Applying the same budget to a system made of even cheaper robots would yield no more than 40 of them and they would lack important communication and localization capabilities. We have not seen research on swarms executing massive random walks, but our estimate is that the triple advantage in number is unlikely to compensate for the inefficiencies due to the lack of planning, especially if 100% coverage is a goal.

3.1 Model Assumptions

The systems and algorithms described in this chapter are based on a set of assumptions about the hardware and the environment on which they will be used. The purpose of this section is to list all of these assumptions and therefore to prevent any potential ambiguity in the following sections.

The localization system is taken to be a black-box function, which when invoked returns the present coordinates of the robot within some small error ϵ. Such a system might be constructed on top of a more erroneous localization system, if errors are uncorrelated or can be distinguished in some other way. In such case, we might wrap a filter around the localization system, which outputs values only when a long enough sequence of consecutive identical values has been received by the black-box hardware system.

The localization system is not bounded in terms of how much time it takes to return a value. However, it is assumed that there is some finite expected delay of response d_L. An obvious corollary of this assumption is that the probability of the localization system taking infinite time to respond is zero, unless the robot fails.

On top of that black-box localization system we assume that there exists a feedback-controlled navigation system. Its assumed properties are similar to those of the localization system. The navigation system is a black-box that allows the robot to go to a point (X, Y) on the coordinate system, if there is a clear straight path between the present position of the robot and (X, Y). Clear straight path is defined as obstacle-free stripe of width w, stretching between the two points.

Just like the localization system, the navigation system is not bounded in terms of the time it takes to complete the operation. However, it is assumed that the average velocity of the operation (that is the distance traveled divided by the time taken by the transition) has a non-zero expectation v_N (which is related to d_L, as the navigation system uses the localization system for its feedback loop). Hence there is a finite expectation $d_N = 1 \ / \ v_N$ of the time it takes the navigation system to take the robot to a specified location a unit distance away. Analogously, the navigation system takes forever with probability zero, unless the robot fails. In addition to all this, we also assume that the additional delay above d_N is memoryless (i.e., the length of the delay up to a given moment is independent of whether the delay will continue or will be discontinued in the next moment).

As mentioned in the introduction, the scanning primitive is abstracted away from the system. That is why, for simplicity of analysis it is considered to take negligible time. Part of the rationale for this is that if the scanning primitive is longer in time than the navigation process, the performance bottleneck of the whole process would be the executions of the scanning primitive, which would mean that how we organize the motion of the robots is of little importance to the performance of the whole system. If we had such a scenario, then the best strategy would be to explore the whole world and then just split the free space equally among the robots and let them search it on a second pass.

Another assumption in this system is that the world consists of two types of terrain: free space and obstacles. The obstacles are those areas not reachable by the robots and so they are excluded from the area that is to be covered. For simplicity of analysis we consider the world to be stationary, that is, every point in the world will always be either free space or an obstacle. Nevertheless, the system should be fully functional in a dynamic world, as long as we consider those areas that change between the two states as being obstacles (that is, they are not required to be covered by the robots at the end). Also, in order to allow the robots to search all of the free space, we assume that the free space is connected and the connections are never too narrow. The latter is defined by specifying that a robot, which is twice as large as the system's robots (twice linearly, or four times in area) is able to reach all of the free space within the scanning primitive radius r, independent of where that robot starts.

Our robots are assumed to be cheap, so no environmental sensors are assumed (including obstacle detectors). However, since obstacles prevent robots from going over them, the navigation system is assumed to be able to realize when it cannot proceed forward to its destination and then report an obstacle in the way. Analogously to the above cases, we assume that the time to detect this situation is unbounded, but with a finite expected value. For simplicity, we take this expected value to be d_N again, but this simplification is not essential to the correctness of the algorithm (it simply allows us to talk about average time to cover a unit of area one unit of distance away, independent of whether the unit eventually turns out to be free space or an obstacle; otherwise we would have to have a formula for that average, which includes the two values and the ratio of obstacles to free spaces the robots encounter). An important distinction of the obstacle detection subsystem is that it should never return false positives, as then the correctness of the operation would be flawed, since the robots would consider some free space, which potentially might contain the target object, as an obstacle.

By this time it should be clear that the motion of the robots is completely asynchronous, which means that predicting which robot will move next (or which robot will reach its destination next) is impossible. In accordance with this, the communication system of the robots is also taken to be completely asynchronous. It allows every robot to broadcast a message at any random point in time, however, the reception of this message by any of the other robots is considered unreliable. When a robot A sends a message, each one of robots B and C might or might not receive it and whether one of them receives it is independent of whether the other (or anyone else) receives it as well. The time taken for a message to be received is unbounded. However, our protocols and algorithms are designed with the assumption that for most of the time the communication system will be successful in delivering messages instantaneously (i.e., in much less time than d_L or d_N). If this assumption does not hold, the protocols and algorithms will still be correct, but they are likely to perform badly. Finally, the communication system is assumed to deliver messages from the same sender in the order of their transmission and to always deliver the messages correctly, without errors (if it delivers them at all). Of course, such a system might be built on top of an error-prone communication system by employing CRC codes or any other error-detection scheme.

The size of the messages in the communication system is assumed to be bounded by some constant s_M, which means that the communication protocols are designed in a way that is suitable for a wide range of possible values of s_M.

In addition to the above, the robots are assumed to be able to distinguish a collision with another robot from a collision with an obstacle (otherwise the robots would be in danger of detecting obstacles where there are no such). One way to achieve this property, without needing a special sensor, would be to assume the communication system is always able to deliver a short message to all robots in the immediate vicinity.

Finally, our failure model is a crash/stopping failures one. We assume that when a robot is broken it stops functioning and never starts behaving in a Byzantine (i.e., arbitrarily malicious) manner. In fact, our system might tolerate partial failures, such as malfunctioning communication and/or navigation systems, but for simplicity of analysis we assume that if a robot breaks, then it stops influencing other robots' decisions.

Since all subsystems are assumed to take unbounded amount of time, one robot can technically fail and then recover (e.g. if it gets restarted) and this would appear as a non-failure to the other robots (i.e., it would be considered simply a long delay). However, it is important that should such an event happen, the robot should either keep its state from before the crash, or it should restart itself with a different unique ID. Otherwise, the robot would be risking breaking some data structure invariants defined in section 3.4.1 below.

3.2 Environment Representation

Since the robots have access to reliable location measurements it is possible to discretize the representation of the world without compromising accuracy or accumulating error. Because of this, and because of preference for simplicity, the robots hold a discrete, rectangular grid representation of the world around them. Every point in the given area is represented by a square, axis-aligned grid cell with a side length of r. Thus, whenever a robot reaches (roughly) the center of a grid cell, it can apply the scanning primitive and search through all the points in that grid cell. The robots will always execute the scanning primitive at, or very close to, grid cell centers. A point is not considered covered unless a robot goes to that point's respective grid cell. The cells are the same for all robots (i.e., all robots are initially programmed to have the same coordinates, offsets and side lengths for all grid cells).

This above paragraph directly implies that the grid cells are of two types: Free and Obstacle, depending on whether their center can be reached. If there is free space in cells, which have obstacles at their centers, we consider this free space covered by the scanning primitive executed at neighboring grid cells (because due to the grid cell side length being equal to r, the scanning primitive clearly extends as far as the centers of the neighboring cells).

This representation allows us to think of the area as a two-dimensional array of boolean variables T, denoting whether the corresponding cell is Free (true) or Obstacle (false). This array, of course, is unknown to the robots initially. Even the boundaries of T are unknown, as the robots do not know the size of the area to be covered in advance. This lack of knowledge prompts the introduction of another array of boolean variables (with a one-to-one relationship with the grid cells, just like the first array). This array, named K, will denote whether a cell's type is known to the robots or not. Moreover, since the robots have no sensory hardware to detect obstacles from a distance, the only way the robots can know the type of a grid cell is to have one of them go to the center of the grid cell, or at least try to go there and get blocked by an obstacle on the way[1]. Because of this, and because applying the scanning primitive is considered to take negligible time, K tells us, for every Free cell, whether it has been covered or not. Consequently, this means that the termination condition of our Area Search process is that K should be true for every cell for which T is true. However, since the robots do not know all values in T, the only way they can be sure the termination condition holds is to observe that in addition to the above, every cell for which T is true has no neighbor[2] cells for which K is false (as then these neighbors might have a true T and a false K, which would break the termination condition, as they would be part of the searchable area, which is not searched yet). Only when these two conditions are met will a robot halt and report its task accomplished.

Now, having this discrete structure in place, we can consider each one of the robots as being a state machine. The state of a robot would be the combination of its memory state and its position, represented by the grid cell it is currently on. The

[1]Later on, when the term "covering" is used, it will refer to both of these operations, depending on the type of cell it is applied to.

[2]Each cell is defined to have four neighbors: those which share a side with it.

actions of the robot, that is the state transitions in the state machine, are of three types:

- Broadcasting a message through the communication system.

- Receiving a message through the communication system.

- Moving from one grid cell to one of its adjacent grid cells.

The environment representation described above reduces the task to one in which we only need to design a communication protocol and a command algorithm for initiating the two actions: sending a message and starting a move, while keeping in mind that the results of those actions (delivering the message and completing the move) are asynchronous and can take arbitrarily long periods, which might in turn rearrange the order in which events are initiated and completed (e.g. robot A might start a move before robot B, but robot B might complete the move first). These actions (particularly the completion of a move) update the K arrays of the robots, as well as their partial knowledge of T, and the process completes whenever the above-mentioned termination conditions on K and T are met and all live robots are aware of that fact.

Finally, it should be noted that the rectangular shape of the grid is not of a particular importance. If a more symmetric hexagonal grid representation is chosen instead, the protocols and algorithms described below should be equally correct and performing equally well.

3.3 Goals and Lower Bounds

The goals of this location-aware Area Search system can be summarized as follows:

1. Solve the problem correctly. Every Free grid cell should eventually be visited by at least one robot.

2. Make sure that the problem is still solved correctly, in the event of robot failures.

3. Complete the task in as little time as possible. This should hold both with the upper bound on worst cases and in practice on "average", realistic cases.

In order to be able to calculate the running time of our system, we need to define a few parameters, which capture the complexity of the problem:

- ***N*** is the total number of connected Free cells that are to be covered.

- ***D*** is the diameter of this set of connected Free cells (i.e., the largest distance between any two cells in the set, measured by the number of cells traversed in the shortest path between them).

- ***B*** is the total number of Obstacle cells that have a neighbor counted in N above. Essentially, B is the length of the boundary of the given area, including the boundaries of the internal obstacles.

- ***R*** is the total number of robots that do not fail before the task is accomplished.

- ***C*** is a communication system penalty factor defined as follows. If we take a robot A and take one of its moves B, the indicator variable $G_{A,B}$ specifies whether A would have decided to take the move B in the same direction, had A known the complete state of every robot in the system. That is, if A knowing all the information known to anyone would have chosen a move in direction X, but instead, because the communication system did not deliver some of this information in time, A decided to take a different direction Y, we set $G_{A,B}$ to 1. Otherwise, we set it to 0. Having these indicator variables we define C as the sum of all G variables, or:

  $$C = \sum_{A \in Robots} \sum_{B \in Moves_A} G_{A,B}$$

  Clearly this penalty factor does not depend on the quality of the communication system alone, but also on the exact protocols and algorithms used. If the robots are spatially very distant from one another, then C will be small, as the state of distant robots is unlikely to affect one robot's decisions. On the other hand, if the robots are always keeping together, trying to work "in sync", C is likely to be much larger, as every bit of misinformation might matter. Nevertheless, we are working under the assumption that the communication system rarely fails to deliver critical information, so we are not trying to design an algorithm which minimizes C (as it is assumed to be small anyway). Keeping C in the bound of the running time of our algorithm is important, however, as it will warn that if run on hardware with a poor communication system the algorithm would perform badly.

- One unit of time is defined as the expected time it takes a robot to go from one cell to one of that cell's neighbors. This is equal to $d_N \times r$.

Having specified these variables and units, we can start analyzing the time that different Distributed Area Search algorithms would take. Covering all N Free cells is inevitable. In addition, covering the B Obstacle cells that are adjacent to the Free ones is also inevitable, as the robots cannot distinguish between the two kinds until after actually spending a unit of time to cover them. Having only R robots at hand, the above means that no algorithm can possibly achieve a time better than:

$$\Omega\left(\frac{N+B}{R}\right)$$

In addition, if we are considering a case in which the robots do not start dispersed throughout the area, but are rather concentrated in one place, then we certainly cannot complete the task in less than $\Omega(D)$, as we need that much time to reach the most distant Free cells. Hence in this case our lower bound goes up to:

$$\Omega\left(\frac{N+B}{R}+D\right)$$

These are the only strict lower bounds we could prove (at least in terms of the variables defined at the beginning of the section), but there are two more bounds, which seem intuitive. This, of course, does not mean that they are indeed lower bounds, but at least we have a reasonable confidence that performing better than these bounds is hard.

The first such bound is $\Omega(C/R)$. In a worst case, all of the C instances of missteps due to delayed communication would result in a robot A moving to a cell where another robot B moved recently, but this piece of information had not reached A on time. This would result in C repetitions of cells by robots, which would inevitably result in C/R wasted time, above the strict lower bound.

Technically speaking $\Omega(C/R)$ is not a strict lower bound because hypothetically one could come up with an algorithm, which makes sure that robots avoid stepping on cells, which were about to be stepped upon by another robot. Coming up with an effective algorithm of this kind is hard, however, for the following reason. Consider the situation from the above paragraph, where robot A is considering covering a cell, which robot B would cover soon. In an asynchronous system with unbounded delays robot B might be experiencing a long delay, or might have failed and so it is reasonable to expect robot A to take over B's "responsibilities". Otherwise, A risks leaving a "hole" in the area of covered cells (that is a solitary cell with a false value in K). If the worse scenario comes into being, this would result in the other robots having to go back and cover this "hole", which might result in a time penalty on the order of D, which would be much worse than the potential waste from covering a cell twice.

The other intuitive bound is $\Omega(B)$ and the intuition behind it is the following. If we have an algorithm that traverses the boundary before the interior of the area, then this algorithm clearly takes $\Omega(B)$ time, especially in a case with concentrated initial positions of the robots, as traversing a boundary is hardly parallelizable. If we have an algorithm, which covers the interior before finding out where the boundary is, then at some point in the algorithm there could be two unexplored corridor-shaped areas separated by a distance of d. The area with more robots can turn out to be deeper, but narrower and as a result these at least $R/2$ robots will cover their area before the other group and will then have to travel the whole distance d to the other group (as otherwise the first group would risk wasting enormous time doing nothing). When the two groups finally join, they might find themselves in the same situation: with two corridor-shaped branches with a total depth of d. Then again the much longer

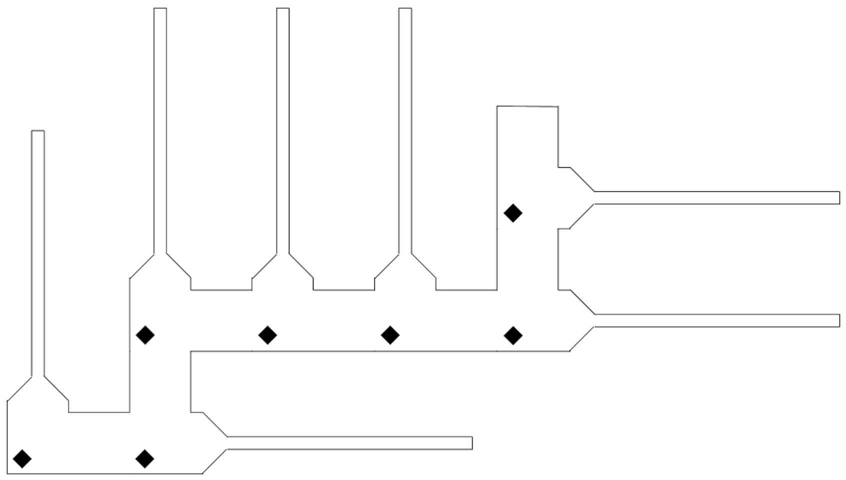

Figure 3-1: A case suggesting an $\Omega(B)$ lower bound. The robots start at the lower left corner and rejoin on every diamond.

one will be covered first and the story will repeat again and again. This would give us a worst case with $D = d$ and $B >> d$ in which half the robots spend $\Omega(B)$ time traveling from one spot to another over already covered cells. See figure 3-1 for an illustration of a case like this.

Of course, the above is not a strict proof of $\Omega(B)$ being a lower bound on the time needed to solve Distributed Area Search. Hypothetically, to the best of our knowledge, there might exist an algorithm that does better. In fact, if the robots knew the map in advance, they are likely to be able to outperform this bound, as then they can plan a more balanced exploration of the area, which does not fall into worst cases such as the one above. Nevertheless, every map-oblivious algorithm we could come up with needed $\Omega(B)$ time in worst cases like the above, which is why we have reasonable confidence that this is indeed a lower bound for this problem.

Based on the above intuition for the lower bounds, we decided to set our goal for the upper bound run time of our algorithm to:

$$O\left(\frac{N+C}{R} + B\right)$$

In fact, since good performance on "average" cases is a requirement and since on most realistic cases the N/R factor is the dominating one, we enforced an even tighter bound on the algorithm:

$$\frac{N}{R} + O\left(\frac{C}{R} + B\right)$$

An easier way to think about these bounds is the following. If we consider all the movements taken by the robots, their number is R times the run time of the algorithm (as one robot accomplishes one movement in one time unit on average). We can break down these movements into two categories: first-time coverings and repetitions. The number of first-time coverings is exactly $N + B$ independent of which algorithm we use, as long as it is correct. Our goal is to bound the number of repetitions by $O(C + B \times R)$. Dividing the sum of these two numbers by R gives us exactly the time bound above.

3.4 Communication Protocol

The communication system is assumed to work correctly and instantaneously most of the time, but in order for this assumption to be realistic we need to have a communication protocol that scales without any dependencies on the unbounded parameters of the problem. In our case, the unbounded parameters are N, D and B as defined in the previous section, which are all properties of the initially unknown map. The other parameter, the number of robots R, is said to be bounded by about 30 in the beginning of this chapter. Hence our communication system should scale reasonably well with the number of robots, but its performance need not be independent of this number.

The objective of the communication system is to make sure that every robot has all the information it needs every time it makes a decision on which direction to take. A requirement for the communication protocol (in addition to the scalability mentioned above) is to be fully asynchronous: no robot should assume the liveness or responsiveness of any other robot at any point in time.

The above-mentioned information consists of the values of T and K learned by other robots, plus optionally some variables defining the state of the other robots. Since for most reasonable algorithms (including the one described in the next chapter) the latter state variables are very few in number, we assume that the state of a robot, which other robots need to know (except for T and K) can be encapsulated in one variable of size s bytes, with s being a small constant.

Since the size of the T and K arrays is proportional to N, it is obvious that transmitting this data repeatedly would break the scalability requirement of the communication protocol. That is why we need a protocol, which transmits only the changes to these data and which transmits every such change at most $O(R)$ times in total.

3.4.1 Data Structures

In order to enable a communication protocol meeting the above requirements, we have decided to organize the state in the following way. Every robot A is the owner of two data structures:

- A's state variable S_A of size s and its version number V_A, which is initialized to zero.

- A chronologically ordered list F_A of length L_A, which contains all values of T and K that A discovered by covering grid cells itself.

The robot which owns a data structure has the only authoritative copy and is the only robot allowed to write to that copy. Because of this, there is a strict order of the elements of the F lists. Also, the definition of the F lists implies that they are only growing (as the robot only discovers new data and never "forgets" what it knew). This means that an older version of an F list is always a prefix of its most recent, authoritative version.

Whenever a robot A changes the data in its owned data structures it follows the following discipline:

- Whenever S_A is changed, its version number V_A is incremented by one.

- Whenever a new value is added to F_A, its length L_A is incremented by one.

Every robot seeks to keep up-to-date copies of the two data structures of all R robots. Consequently, every robot makes sure to announce any new information it has. This is achieved through the usage of the message scheme described below.

3.4.2 Messages

The communication system of each robot sends two types of messages. Type A messages are broadcast repeatedly, for example once every second or some other time period, depending on the particular communication hardware used. Type B messages are only sent in response to type A messages.

Type A messages consist of all V and L values known by a robot (i.e., the V and L values of every robot, as contained in the message sender's copy of those data structures).

Whenever a robot X receives a type A message from robot Y, X compares the V values with its own copies of these V values and the received L values with its copies of these L values. If some of the received values are smaller than those in X's copies, this means that X possesses some information unknown to Y. If that is the case, X sends a type B message to Y, which contains the following:

- For every V_i, which is smaller than X's copy of V_i, X sends to Y its copies of V_i and S_i.
- For every L_i, which is smaller than X's copy of L_i, X sends to Y all elements of X's copy of F_i, which have indices higher than Y's version of L_i.

This type B message essentially contains all the state information known to X, which is unknown to Y (or at least which was unknown to Y at the time it sent its type A message). Of course, if the data to be sent in this type B message is larger than the maximum packet of the communication system, X can divide the type B message into several smaller ones, each containing some part of the data. However, X should make sure that F values with lower indices are sent before F values with higher indices, as presently Y has no mechanism for keeping disconnected fractions of an F list in memory.

Whenever a robot receives a type B message, it updates its data structures with the new information:

- If a received V_i value is higher than the local copy, then the local copies of V_i and its corresponding S_i are replaced with the new values.

- If a value of F_i with an index one higher than the local copy of L_i is received, then this value is appended to the local F_i list and the local copy of L_i is incremented by one.

The communication protocol described above ensures that every new piece of information eventually reaches every robot. If two robots can communicate with one another, it takes only two steps (a type A message and a type B message) for the information known by the first robot to be delivered to the second robot. In fact, if the communication system is perfectly functional at the moment, and if all sent messages are broadcast, every new piece of information will be delivered to all robots in just two steps. Also, if not all robots are directly connected to everyone else, the communication system will "gossip" the new information to everyone, as long as the whole network is connected. Finally, if the robots split into two (or more) groups for a while and then connect again, the communication protocol will make sure all the information known by the first group is delivered to the second group and vice versa.

The timing of the above operations can be improved if we add some optimizations to the protocol. Here are some examples:

- Whenever a robot augments its F list or its S state, it broadcasts a type B message with this information. This will make dissipating new information a single-step process in the error-free case.

- A robot might append its most recently learned F, S and V values (normally sent over type B messages) to its type A messages. This will reduce the number of messages sent when "gossiping".

Finally, it should be noted that this protocol is fully asynchronous (i.e., no robot ever waits for response from another robot) and meets the scaling requirements.

Namely, the load on the communication system increases only with the increase of R, but not with increasing N, D or B.

3.5 Compact Coverage Algorithm

Having defined a communication protocol, which meets the requirements of the assumptions given in section 3.1, we can proceed with a state-machine algorithm that assumes all the needed state information is available to the robot at the time of the decision.

If this assumption does not hold and our algorithm produces undesired output (e.g. it sends the robot in a wrong direction), we know that the cost of this, in terms of needless repetitions of covered grid cells, is proportional to the duration of the misinformed state. The reason is that for every misinformed decision the robot drifts at most one cell away and so returning and then taking the right course takes no more steps than it took to drift away (i.e., no more steps than the duration of the misinformed state). This guarantees us that the total cost of the cases in which our assumption does not hold is $O(C)$, which when measured in time is $O(C/R)$, which is included in our target upper bound as described in section 3.3.

Now, all we need to do to solve Distributed Area Search in the desired time bound is to come up with a state-machine algorithm, which assumes no communication problems (i.e., it can safely assume all state is shared by all robots instantaneously) and has each robot making no more than $O(B)$ repetitions of covered cells.

A number of different approaches were considered. Many of them were based on an idea presented by Burgard et al. [2] stating that trying to keep the robots away from each other will help them avoid repeating one another's work and will result in better performance. While these algorithms performed very well in practice, it was very hard to prove a worst-case bound on their performance. In fact, for most of them we were able to construct a worst-case scenario that breaks the desired bound.

Another set of algorithms we considered attempts to divide the area into "convex" regions and then have the robots cover each such region with a very simple divide-

and-conquer algorithm. However, these algorithms were very sensitive to single-cell obstacles, and thus, they required a certain level of synchrony between the robots, which is infeasible under our model.

Yet another approach considered was trying to construct a Hamiltonian cycle over the grid cells' adjacency graph, or an Eulerian cycle over the obstacles' visibility graph. The latter resulted in a nice algorithm with a very strict proof of the complexity bound. Unfortunately, the practical performance was very bad, as the algorithm almost always performed as badly as in a worst-case scenario. While this can probably be resolved, doing so would complicate the algorithm too much.

Finally, we decided on an approach in which the robots keep a connected, well-shaped set of covered cells and expand it relatively uniformly, without leaving "holes" behind. We propose an algorithm called the Compact Coverage Algorithm (CCA) based on this approach.

An intuitive description of the CCA goes as follows. At any point in time the robots have covered a connected set of cells (i.e., one can go from any covered cell to any other covered cell by walking only over covered cells). Each one of the robots is working on the front line of this covered set (i.e., robots always cover uncovered cells, which are adjacent to covered cells). The direction in which they walk along the front line, clockwise or counter-clockwise[3], is kept constant for every robot until that robot meets another robot on the front line. Whenever that happens, the robot turns around (changes from clockwise to counter-clockwise or vice versa) and keeps covering the front line, this time in the other direction. In addition, whenever the robots split the uncovered part of the area into disjoint regions, they make sure to traverse these regions in an orderly, DFS fashion, not in random order.

Figures 3-3 to 3-9 demonstrate an example run of the algorithm on a square area with four square pillars in the middle. On all figures covered Free cells are in white, covered Obstacle cells are in black, and uncovered cells are in gray. The robots are marked by unique letters.

A formal definition of the algorithm, as well as proofs of its correctness and its

[3]See figure 3-2 for an illustration.

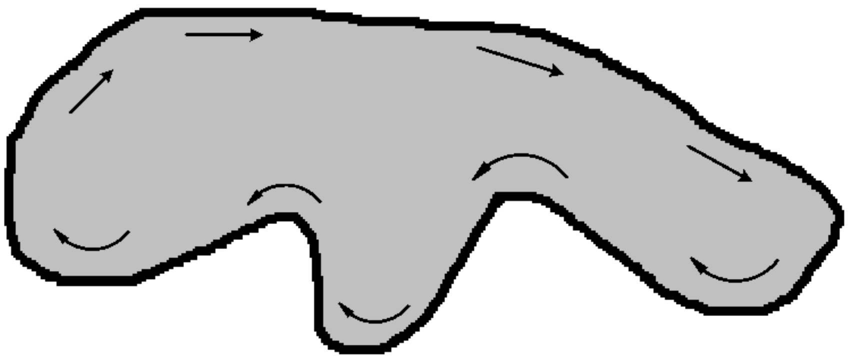
Figure 3-2: An illustration of the meaning of “clockwise” direction on the front line.

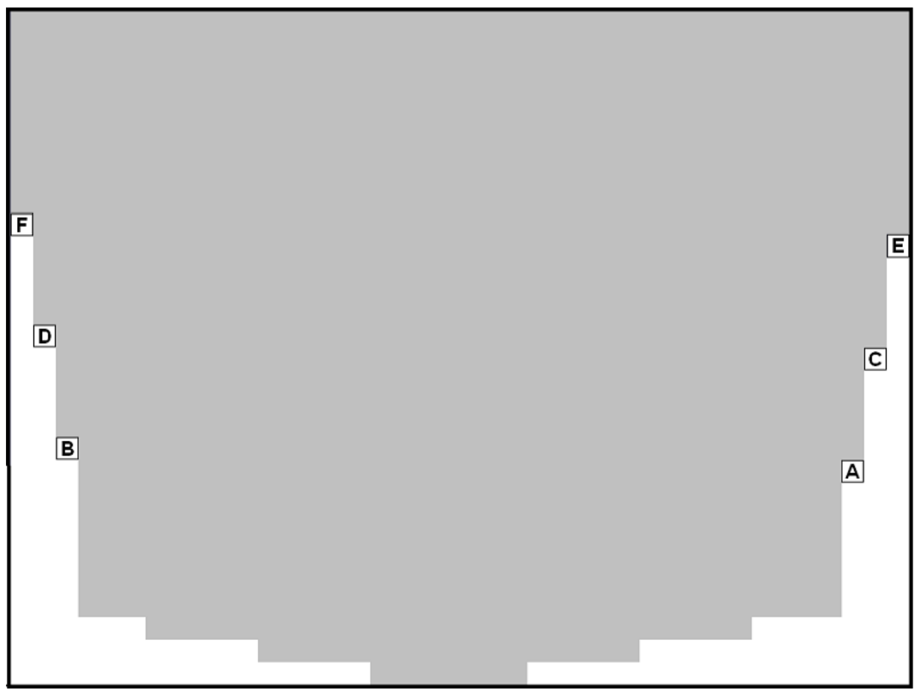

Figure 3-3: Robots F, D and B are covering the area in a clockwise direction, while E, C and A are going counter-clockwise.

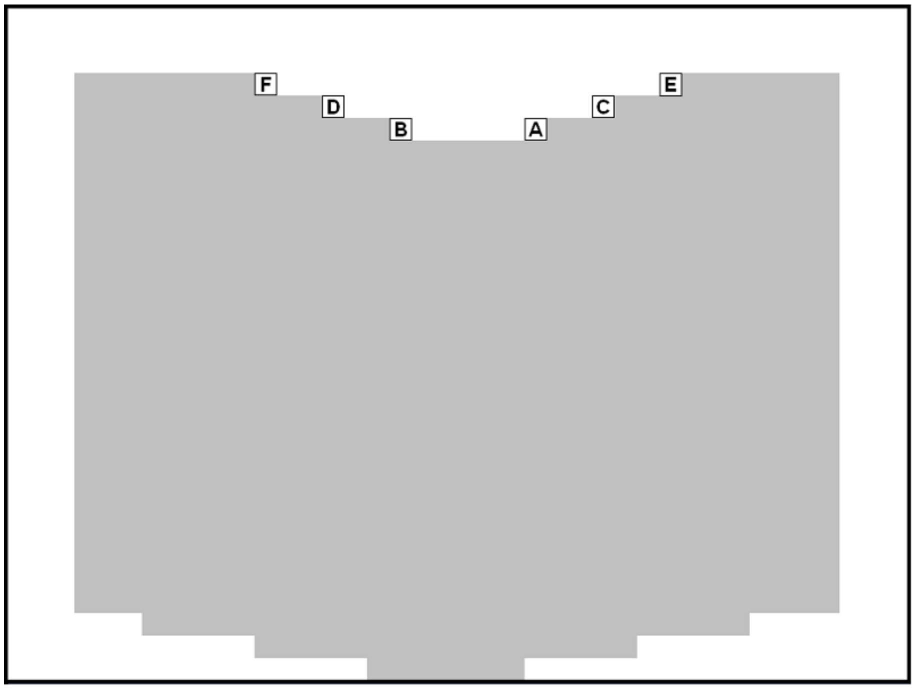

Figure 3-4: The two groups of robots have just met one another and have turned around

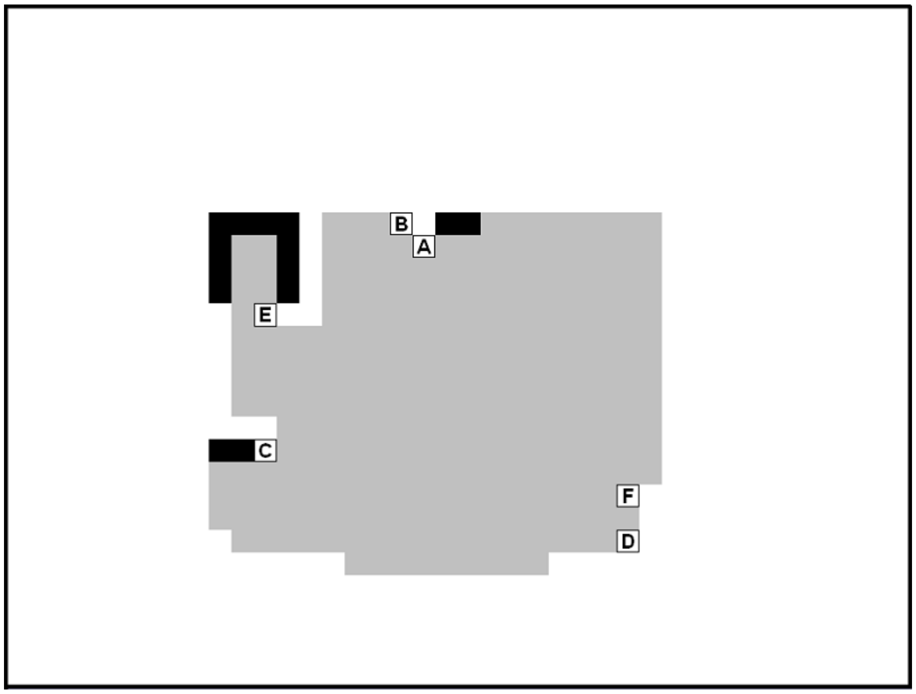

Figure 3-5: The robots are starting to meet obstacles.

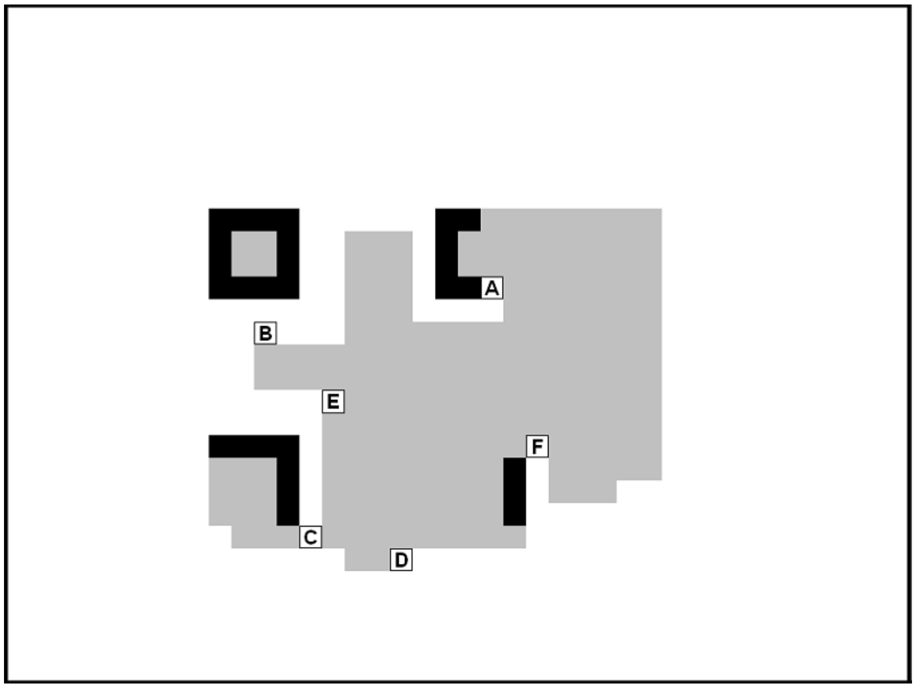

Figure 3-6: Robot C has just split the uncovered region into two separate regions.

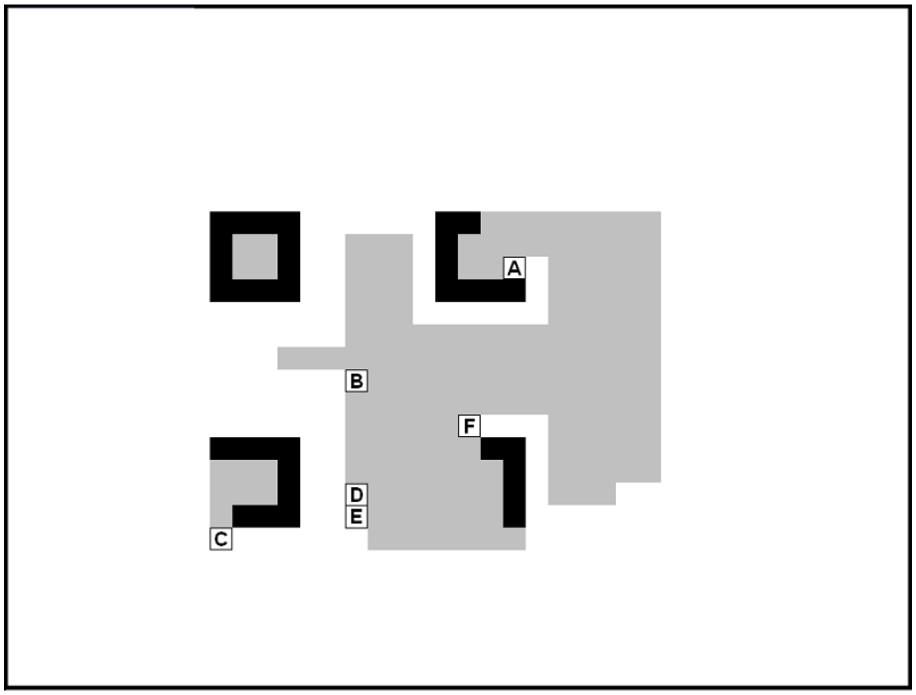

Figure 3-7: Robot C covers the smaller, left region, while the others are working on the much larger, right region.

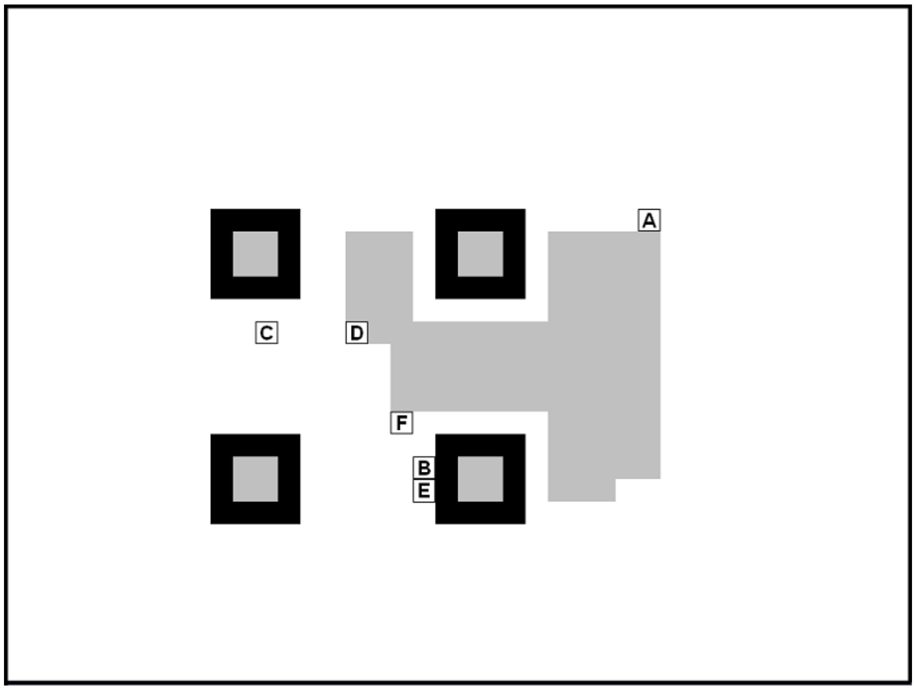

Figure 3-8: All robots are gathering towards the last remaining region. All other regions have been covered.

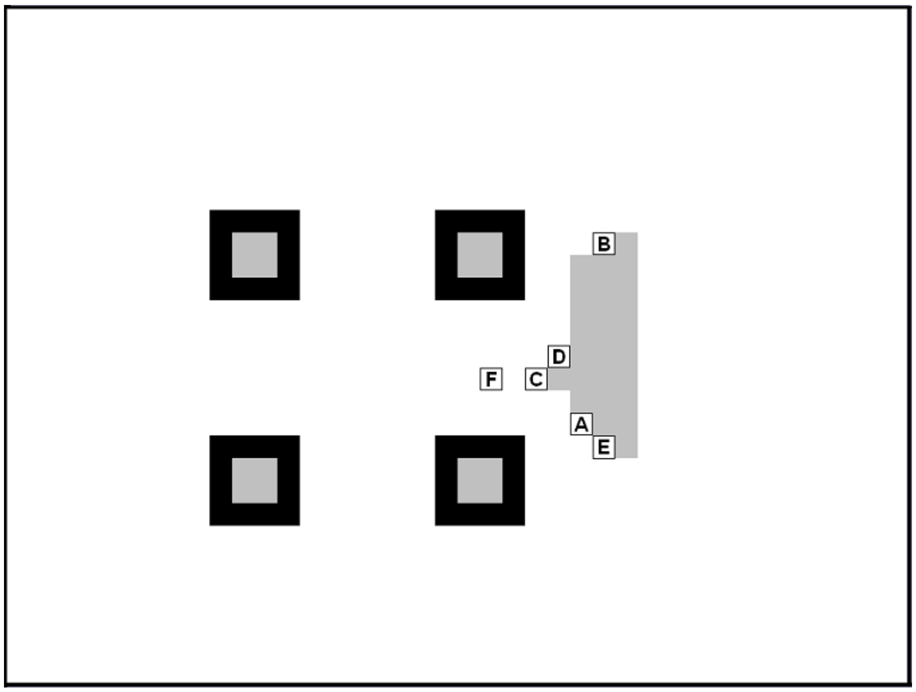

Figure 3-9: The robots are near the completion of their task.

performance bounds are presented in the next subsections. For simplicity of analysis we will assume all robots start nearby, very close to one another, and very close to the border of the area, so that they can have the connected set of covered cells since the very beginning. Nevertheless, the bound can be shown to hold even if the robots start away from one another and/or away from the border.[4]

3.5.1 Formal Definition

In addition to the T and K arrays, the present coordinates of the robot and other trivial variables, or variables already mentioned in section 3.4.1, the state of a robot includes:

- A connected graph tree G, whose nodes are all connected regions of unexplored cells. These regions consist of cells with false K values, which are connected in the sense that one can go between any two cells in that region by walking only over other members of the region. G is essentially an adjacency graph over these regions, showing which regions were directly split from one another (and thus are now adjacent to one another). Travelling between adjacent regions involves walking only over former (now covered) cells of these regions. The edges of G are grid cells. The edge between vertices A and B is the cell, which (when it was covered) first separated A and B into two regions disconnected from one another (i.e., this is the only cell on the path between A and B, which belonged neither to A nor to B, but belonged only to the parent from which A and B split). Initially G contains only one vertex.

- A list VG of the G nodes that have been visited by the robot. Initially this list contains only the single vertex of G.

- A boolean variable CW specifying whether the robot is covering the front line in the clockwise (true) or counter-clockwise (false) direction. This is defined by the direction of the loop, which the robot will complete in case it continues on

[4]In fact, in practice the robots perform better if they start more evenly dispersed throughout the area, so having them starting close together is somewhat of a worst case.

the front line without interruption. For example, if we look at a 2-D map of the area (with up, down, left and right directions) and if the covered cells are below the robot and the uncovered cells above the robot, then the clockwise direction is to the left, as the robot will surround the uncovered cells by going left, then up, then right, then down, then again left, eventually completing a clockwise-directed loop. See figure 3-2 for an illustration of this example. The initial value of CW is arbitrary.

In addition to the communication operations described in section 3.4.2, the robots also perform movement actions. These actions are performed when the previous movement action completes. This means that the present grid cell of the robot is covered and also that at least one of its adjacent cells is also covered. The algorithm for deciding the direction of the next move is the following:

- If the cell just covered by the robot split an uncovered region into two regions[5], then the robot should update G accordingly. The old node should be split into two new nodes. If the current cell turned out to be Free, then it should become the edge connecting the two new nodes. If the just-covered cell was an Obstacle, then we should connect the two new nodes with another cell (a Free cell) that they shared in common before the split occurred. If there is no such cell, then there should be no direct edge between the two nodes in G. Also, the edges coming into the old node should be distributed among the new nodes according to where these edge-cells are (e.g. if the region is split into an eastern and a western half, then all eastern edges should be connected to the node representing the eastern half).

  After the old node-region is split, the robot has to pick one of the new node-regions as its destination and move to one of the adjacent cells that belong to that region. If the just-covered cell was an Obstacle, then the robot should remain in the region on the same side of the obstacle as the robot. Otherwise both regions are an option and picking any of them would be correct. In practice,

[5]If the split separated more than two regions, then we can think of it as several consecutive splits in two.

however, better performance is observed if the robot goes to the region farther from the return path of the DFS over the graph. In order to pick that node, the robot should look at the VG list and find the node X, from which the robot first came to the old node. The robot will eventually have to return to X, so it is better to pick a new node Y, which is away from X (as opposed to the new node Z, which is connected to X in G). This way, the robot will do a sequence Y-Z-X, instead of the longer Z-Y-Z-X. Both are valid DFS paths though, so either choice would fit into the bound (as proven in the next subsection).

- When a robot is on the front line, on the edge between the covered region and an uncovered region (i.e., on a just-covered cell, which has adjacent uncovered cells), the robot should pick its next destination in accordance with its CW variable. Without loss of generality, let us assume that the last move of the robot was in the upward direction[6] (which guarantees that its downward neighbor is a covered cell). Then if CW is true (a clockwise direction), the aim of the robot will be to keep the covered region border to its left. Hence the algorithm will check the left, upward and right neighbors in this order and direct the robot to the first (in this order) available cell. If the CW variable is false, then the order will be reversed: right, upward, left. This will ensure that the robot stays on the front line, keeping the border between covered and uncovered cells on one of its sides, either left or right, depending on CW.

- In the above case, if our robot decides to go to a cell where another robot has already decided to go, our robot should change its mind, invert its CW variable and repeat the above decision with the new CW value. If this does not change our robot's desired direction, then clearly this "contested" cell is the only choice (unless our robot wants to risk leaving "holes" behind) and so this choice should be taken anyway. In addition, if the other robot had a different CW variable than ours, then the other robot should invert its CW variable as well, so that the two CW variables remain different.

[6]The other cases are simply a rotated version of this one.

- In case the robot's cell has no uncovered neighbors, the robot should go to the closest cell belonging to its region (i.e., the last node on the VG list, denoted by LVG). If LVG has had all of its cells covered, then our robot has to pick a new node to go to. Picking the next node is done as if we are executing a DFS algorithm on G, with VG being our history. If LVG has any incomplete (i.e., not yet covered) neighbors, our robot should go to the closest of these neighbors. If there are none, then the robot should backtrack to the node from which it first came to LVG and then repeat the same procedure again with it (i.e., look for incomplete neighbors, if none then backtrack, etc). If there are no nodes left uncovered, then terminate the algorithm and declare the mission accomplished.

3.5.2 Proof of Correctness and Performance Bounds

To prove correctness of the Compact Coverage Algorithm, we need to show that it always covers all Free cells before terminating. As is evident from the last paragraph of the algorithm definition above, a robot cannot decide that the task is accomplished, until after it has completed a DFS traversal of all regions of uncovered cells. The only condition, in which our algorithm might terminate, is one in which there are no reachable uncovered cells (ones with false K). Since all Free cells (those with a true T) are connected, then they must all be reachable. Hence, all cells with a true T have a true K (meaning all Free cells have been covered), which is precisely the definition of the correct termination condition for the Distributed Area Search problem as specified in section 3.2.

Now that we know our algorithm is correct, we can proceed with proving the desired bound on the time that it takes. Sections 3.3 and 3.5 already gave the following bounds:

- The time taken by covering uncovered cells is N/R.

- The time wasted due to communication problems is $O(C/R)$

Hence, the only thing required to prove the desired bound of

$$\frac{N}{R} + O\left(\frac{C}{R} + B\right)$$

is to prove that the time wasted in repeating covered cells is bounded by $O(B)$. This is equivalent to saying that the number of cell repetitions is bounded by $O(B \times R)$, as the R robots are running simultaneously.

In order to prove this bound, we define a helper function $F(X, T, K)$, on a set of cells X, as: the total number of covered and Obstacle cells (i.e., cells with true K or false T), which are adjacent to uncovered Free cells in X (i.e., cells with a false K and a true T, which are in X). The intuitive meaning behind this definition is that $F(X, T, K)$ is equal to the sum of the lengths of the boundaries of the uncovered regions in X, plus the sum of the boundaries of the obstacles inside these uncovered regions. Of course, in most cases the value of this function will be unknown to the robots, as it depends on unknown parts of the world (e.g. the obstacles inside uncovered regions are unknown to the robots). Nevertheless, this function has a clearly defined value at any point in time for any cell set. We can use this function to prove the desired bound on the CCA by proving these two lemmas:

Lemma 1 *If W denotes the whole world, then $F(W, T, K)$ is bounded by $O(B)$ at any point in time.*

Lemma 2 *Any connected region of uncovered cells A can be covered with $O(F(A, T, K) \times R)$ cell repetitions on cells in A.*

Clearly, proving the two lemmas is sufficient, as applying them to the initial moment with $A = W$, we directly get the desired bound of $O(B \times R)$ cell repetitions for covering the whole area.

Proof of Lemma 1

Unless interrupted by obstacles or other robots, whenever a robot covers uncovered cells it keeps walking along the "front line" (i.e., uncovered cells, which have covered neighbors) in the same direction, eventually completing a loop around an uncovered set A. The effect of this loop on the F function of A is always to decrease it by 8. The reason is that every turn inward decreases F by 2, while every turn outward increases it by 2. Since A is internal to the loop, there are a total of 4 more inward turns than outward ones, hence the resulting change of 8. This is also intuitive, as when the A region shrinks it is natural to have its boundary shrinking as well.

This loop can only be interrupted in three ways:

Case 1: The robot is interrupted by an obstacle. In this case, the robot will in fact still be able to complete its loop, even though it will probably take more time (as finding out where the boundary between Obstacle and Free cells is, by trial and error, takes twice as long as just walking along that boundary). The only effect of the obstacle will be that among the neighbors of the uncovered cells, some Obstacle cells will be replaced by covered Free cells. This will not change F however, as it is computed as a sum of these two types of cells.

Case 2: The robot is interrupted by another robot coming in the reverse direction. In this case, the algorithm makes the two robots effectively "switch places" by exchanging their CW variables. This allows the robots to essentially complete each other's loops without having to repeat cells due to the crossing of one another's paths. The effect of this operation on F is null, as there is no difference between two robots completing their respective loops uninterrupted, and the two robots swapping their loops and taking each other's place.

Case 3: The robot X reaches another robot Y, which is walking on the same front line and in the same direction. This is, in fact, the only event that can actually prevent X from keeping its usual course of completing loops and decreasing F. In this case, X will turn around and will leave its loop incomplete, which means that X might have turned outward more times than inward (but the difference would be less than four[7]). This means that the effect of the partial loop on F can be as bad as an increase of six (two additional "front line" cells per three outward turns). Hence, in

[7]Otherwise the robot would have completed a loop around a covered set internal to the uncovered set, which would be impossible under the assumption that the initial covered set is adjacent to the boundary of the world.

order to prove Lemma 1, we need to bound the number of times these partial loops can happen.

The reached robot Y can be one of two kinds: a robot walking around an obstacle while discovering its boundaries, or a robot which simply walks along a non-obstacle front line.

Case 3a: Y is on an obstacle boundary. In this case, Y would normally be slower than X, as walking along such a boundary involves bumping into the obstacle at each one of its borderline cells. This process would slow the traversal of the boundary by a factor of two. If that is the case, however, the number of robots that will eventually reach Y and turn around (just like X) will be at most proportional to the length of the obstacle boundary. Thus, their effect of increase on F will be limited by the length of the boundary. Also, once "used" for this purpose, this boundary cannot be used again, as now it is no longer adjacent to uncovered cells. Hence the total effect of all events of this kind on the whole area would be limited by $O(B)$.

Case 3b: Y is not slowed down by an obstacle. In this case, let us denote the distance between Y and X with Δ. We know that the navigational delays are memoryless[8], and we know that the two robots are travelling along the same front line in the same direction, so the terrain is the same for both. This means that the average probabilities of Δ increasing by one or decreasing by one are the same, and it also means that we can model the value of Δ as a 1-D random walk, as long as it stays within its allowed values[9]. Then, the expected time for X to reach Y is greater than the expected time for a 1-D random walk to return to its origin (since we start with $\Delta > 0$) and we know that this expected time is infinite [8], so for the finite duration of our algorithm, we would expect Y and X to reach one another only a constant number of times.

The 1-D random walk model would be perfectly valid if X and Y were on an infinite "front line". However, they are on a self-connected front line with some finite length FLL. Hence the random walk model would be valid only until the two robots travel

[8]See section 3.1.

[9]See the next paragraph for a more detailed discussion.

FLL steps, and so, we can only safely assume that they meet a constant number of times per FLL moves. However, this is enough for our goal, as one robot completes a loop in FLL steps. The number of loops a robot can complete is bounded by the diameter of the uncovered set, as every loop shortens this diameter. Also, this diameter is never greater than the F function of this set, as the diameter of an area is never greater than the length of its boundary. Hence, the total increase of F that can be due to X reaching Y and vice versa is bounded by F itself, which means that if F was $O(B)$ before this effect, it will continue being $O(B)$ after the effect, as the effect is allowed to happen only once per uncovered set.

The above analysis is easily extendable to R robots walking in the same direction on the same "front line". In this case, we have R pairs of adjacent robots on the line, but we only have FLL/R steps until one robot completes a loop (on average), so the number of cases when one robot reaches another is again bounded by a constant per unit of F for the whole coverage of the uncovered set.

Finally, we should note that the F function evaluated on the whole world W is no greater than the sum of the F functions evaluated on the separate connected sets of uncovered cells. Thus, if the total increase of the F values of these sets is bounded by $O(B)$, as we saw above, then the total increase of $F(W)$ is also bounded by $O(B)$. In addition, the initial value of F is $O(B)$, as initially there are no covered cells and so the whole value of F comes from the obstacle boundaries. Hence, $F(W, T, K)$ is always bounded by $O(B)$, which is precisely what Lemma 1 states.

Proof of Lemma 2

The proof of Lemma 2 is by induction. That is, we will prove that any connected region of uncovered cells A can be covered with $O(F(A, T, K) \times R)$ cell repetitions on cells in A, provided that this statement holds for all proper subsets of A (i.e., any set $B \subset A$ can be covered with $O(F(B, T, K) \times R)$ cell repetitions on cells in B).

The definition of the CCA in the previous subsection shows that robots repeat cells only in two cases. First, when traveling between connected uncovered sets. Second, when covering one cell, which is already picked by another robot, as the only option remaining for not disconnecting from the set.

In the latter case, the mentioned cell can only be a dead-end corner cell[10], and so covering it would reduce the uncovered set to one with a smaller F value (smaller by two). Thus, if we denote the uncovered set, which this cell is part of, by A, and if we denote A without this cell as A', we get $F(A) = F(A') + 2$. Also, the number of times the robots could repeat this cell in this way is $O(R)$. Thus, by the induction hypothesis, the cost (in terms of number of cell repetitions) of covering A is $O(F(A') \times R) + O(R)$, which is equal to $O(F(A) \times R)$, which is the desired bound.

In the other case, when a robot is travelling between uncovered sets, we know that the CCA performs a DFS on the graph of uncovered sets G. In order to prove the desired bound of $O(F(W) \times R)$ all we have to do is show that when doing the DFS, one robot repeats no more than $O(F(A))$ cells per node, where A is the uncovered set corresponding to that node. This is sufficient, because the DFS processes each node only once and so we would get a total number of repetitions by a robot bounded by $O(F(W))$. The cost of traversing one node A in G includes covering the whole node A, entering and returning from all of A's unvisited neighbors and then returning back to the node we originally came to A from. This whole cost, we claim, is bounded by the length of the original boundary of A (i.e., the length of A's boundary, when it was created). The reason is that if a robot B wants to traverse A, all B has to do is stick to A's boundary and it will definitely pass by the "entrance" of (i.e., the cell that is the the edge to) each one of A's neighbors in G. Since the original boundary of A is bounded by F, we get a bound of $O(F)$ for the number of cell repetitions one robot does on a node for all DFS operations together. Hence, the cost from all inter-region travels, accrued by all robots is bounded by $O(F \times R)$ for every region.

Just like in the previous proof, here we also have to mention that the total cost is no greater than the sum of the costs of the separate regions. This is illustrated by the fact that whenever one region splits into two, we can apply the induction principle and conclude that the $O(F \times R)$ bound holds, because it holds for the two smaller regions and their F values sum up to the F value of the old, bigger region.

[10]This is an uncovered cell surrounded by covered cells from at least three sides.

Finally, we have to prove that a DFS traversal of G is indeed possible. We start with the fact that G is connected. All nodes come from the first node in G and since we know all Free cells are connected, this means that even when two nodes are split from the same node, but have no edge between them (e.g. when an uncovered set is explored both from its eastern and its western side and eventually it turns out the two are two separate "rooms" without any direct link between them), there is still a path between them through the other nodes in G.

In order to be able to do a DFS on a graph, we must be able to walk into a node, then walk into its unvisited neighbors and also cover the node either on the way forward or on the way back. What should never happen is to have additional edges attach to that node after we have returned from it (as then we would have to go to that node for a second time). Fortunately, this can never happen under our algorithm, as edges are added to the nodes only when we split a node into new nodes. The edge between the new nodes is the only new edge, since all other edges were only moved from the old node to one of the new ones. Since in order to split a node in two a robot must be present on the node, it means that this robot would not have the above-mentioned problem with performing the DFS. Moreover, no other robot would have that problem, as according to the CCA definition a robot does not leave a node until the corresponding region has been covered completely. Hence, if a robot has exited this node, it means that there will be no new edges attached to that node, which guarantees that a DFS traversal can and will be performed by every robot (except when a robot does not visit a node because the other robots have already covered it completely).

This concludes the proof of the two lemmas, which as shown above are sufficient to prove an expected running time of our algorithm of

$$\frac{N}{R} + O\left(\frac{C}{R} + B\right)$$

with the variance being only over the random delays of the localization and navigation systems and not on the size and/or shape of the area to be covered.

3.6 Simulation Comparison

In order to test the scalability of our algorithm, and also to verify the bounds proved in the previous section, we implemented a Java simulation of the CCA. The simulation only evaluated the discrete Compact Coverage Algorithm and did not attempt to simulate the continuous real-world environment.[11]

In addition to the CCA, we implemented a simple control algorithm, for the purpose of comparing the CCA with a basic, intuitive algorithm and evaluating the improvement achieved by the CCA. The control algorithm consists of three simple steps:

1. If a robot can keep going forward, without running into another robot, and still keep covering uncovered cells, then the robot keeps going forward.

2. If a robot cannot do the above, but it can turn (either 90, 180 or 270 degrees) and then be able to do it, then the robot turns to the smallest angle (in the above order) that allows it to execute step 1.

3. If a robot cannot do the above either, then it goes to the closest uncovered cell, taking the shortest path over covered cells.

This algorithm is both simple enough and good enough to serve well as a control algorithm. There are no obvious improvements to it, and in fact any reasonable behavior that can be expected from a Distributed Area Search algorithm, such as covering an uncovered rectangle optimally, is achieved by this algorithm.

A goal of the simulation is to be as realistic as possible, and for that purpose the tests on which we compared the two algorithms were all taken from real-world environments. The first test is part of the floor plan of the second floor of the MIT Stata Center, home to the MIT Computer Science and Artificial Intelligence Laboratory.[12] This test is characterized by a combination of wide-open areas and narrow corridors. The second test is part of the Caribbean Sea with the many islands

[11]See chapter 4 for a real-world evaluation.

[12]In fact, this is an old version of this floor plan. The actual floor plan is slightly different, but the features are nevertheless the same.

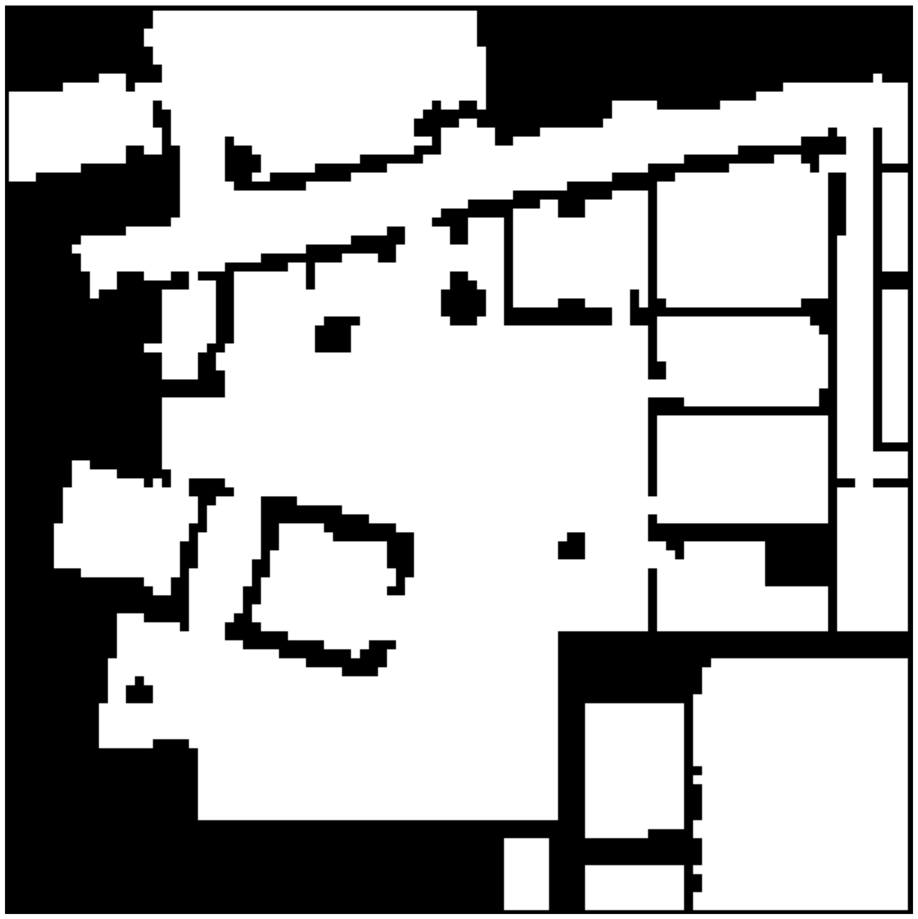

Figure 3-10: Test case 1: Floor plan in the Stata Center.

serving as obstacles. Wide-open areas are clearly prevalent in this case. The third test is a recreational park plan with the trees, bushes and benches serving as obstacles and the grasslands as free areas. The distinguished feature of this case is the large number of middle-sized obstacles. The fourth and last test is a living room interior design, which features mostly large, but few in number obstacles.

The results of the simulation runs are presented in table 3.1. The three central columns show the time taken by a theoretically optimal algorithm and by the two algorithms we evaluate. The unit of time, as discussed in section 3.3, is the average time taken by a robot to go from one cell to an adjacent cell.

Clearly, the Compact Coverage Algorithm performs much better than the control algorithm. In fact, the CCA is even close to the theoretical optimum. The last two columns of the table show that sometimes the CCA outperforms the control

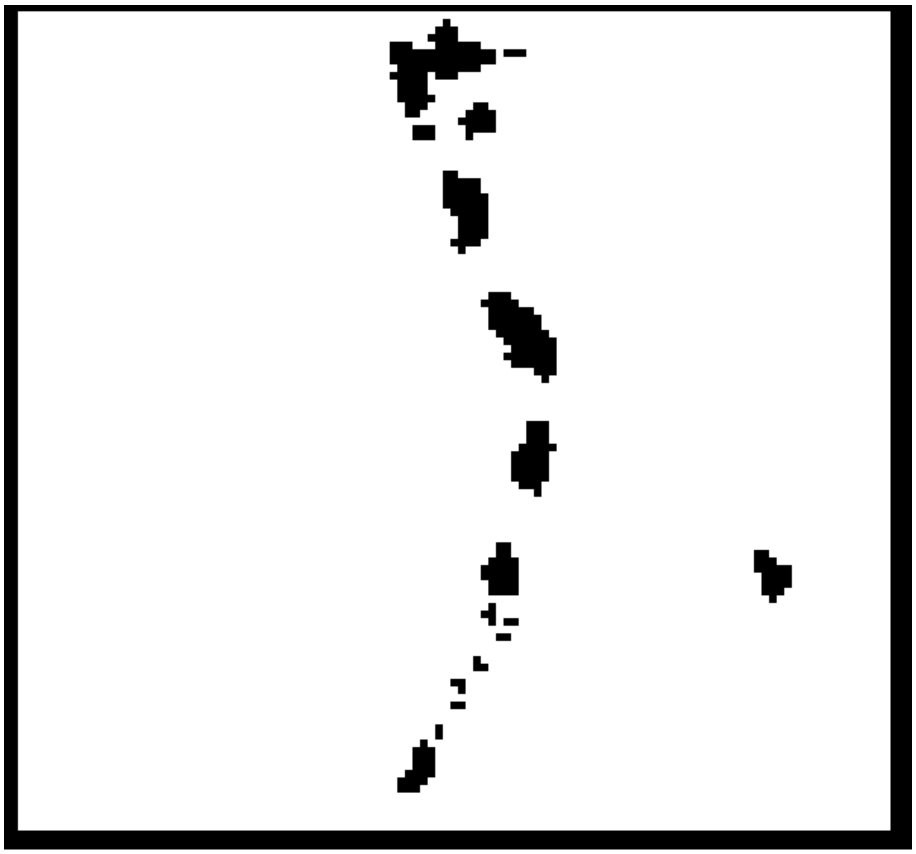

Figure 3-11: Test case 2: Part of the Caribbean Sea.

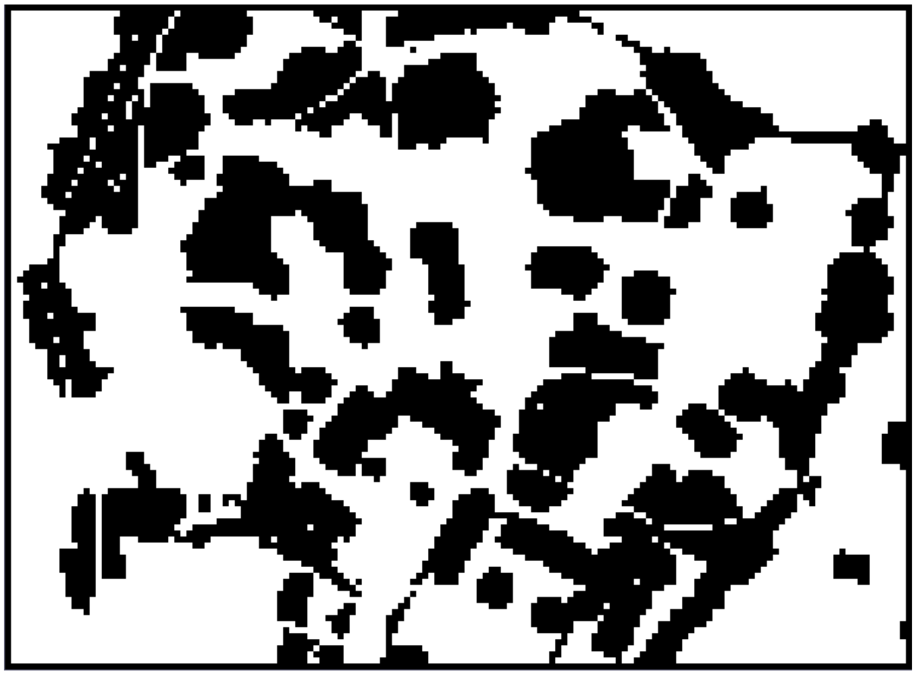

Figure 3-12: Test case 3: A recreational park plan.

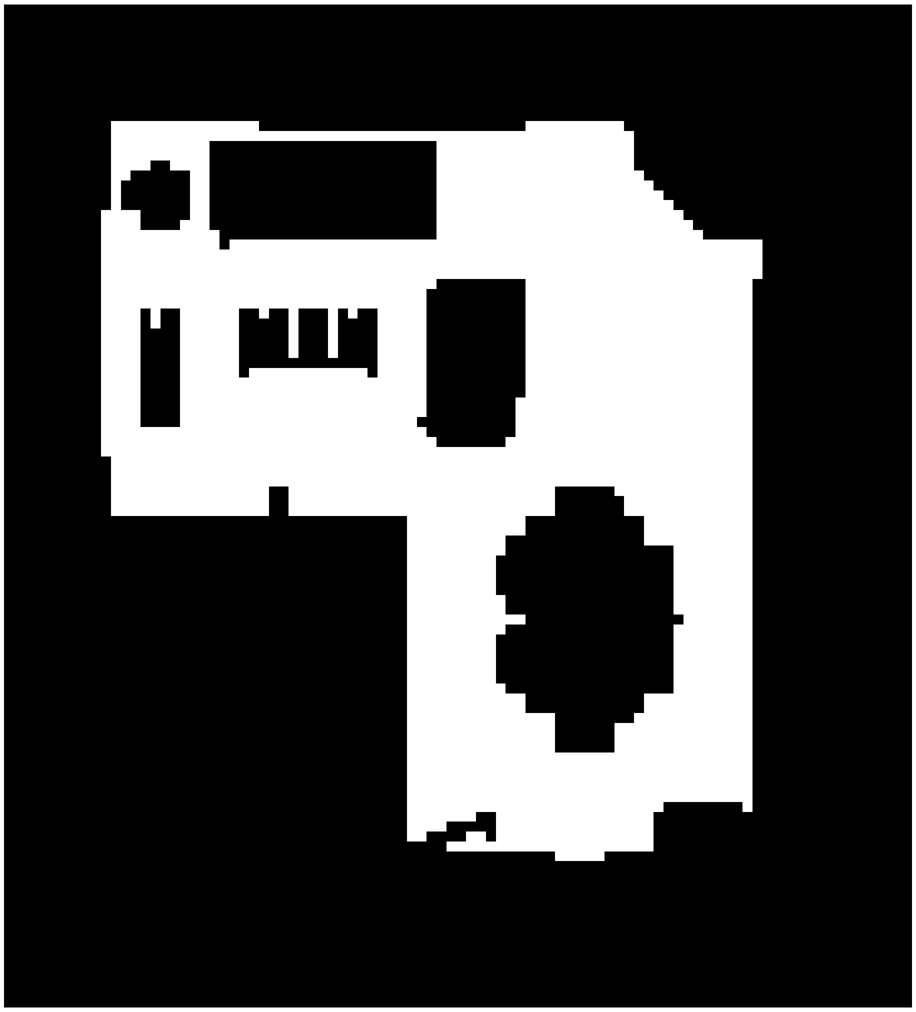

Figure 3-13: Test case 4: A living room.

Test	N	B	R	Theoretical Optimum	Control Algorithm	CCA	$\frac{CCA}{OPT}$	$\frac{Control}{CCA}$
Stata Center	5651	1013	1	6664	8611	7137	1.07	1.21
			2	3332	5538	3708	1.11	1.49
			5	1333	2202	1613	1.21	1.37
			10	667	1907	903	1.35	2.11
			20	334	1267	634	1.90	2.00
Caribbean Sea	12077	641	1	12718	13808	13214	1.04	1.04
			2	6359	8254	6585	1.04	1.25
			5	2544	3607	2740	1.08	1.32
			10	1272	2013	1444	1.14	1.39
			20	636	1061	802	1.26	1.32
Park Plan	8702	1981	1	10683	14269	11828	1.11	1.21
			2	5342	7767	6059	1.13	1.28
			5	2137	4108	2580	1.21	1.59
			10	1069	2790	1388	1.30	2.01
			20	535	1642	848	1.59	1.94
Living Room	2688	566	1	3254	4067	3526	1.10	1.13
			2	1627	2063	1842	1.13	1.12
			5	651	1146	776	1.19	1.48
			10	326	1017	435	1.33	2.34
			20	163	575	303	1.86	1.90

Table 3.1: Simulation results.

algorithm by a factor of 2 or more, while at the same time it is never outperformed by the theoretical optimum by a factor of 2 or more.

Another interesting observation is that the CCA outperforms the control algorithm the most (in terms of the ratio of the time they take), when they are run with $R = 10$ robots. We can better understand this phenomenon if we think about the extreme cases in R. When we run the system with one robot only, most of the time is spent on covering uncovered cells and any reasonable algorithm spends a small percentage of its time repeating cells. Even on the third test case, when the difference between the times of the two algorithms is substantial, the ratio is still not very different from 1. In the other extreme, if we take a really large R, then the fundamental limits of the area (such as narrow corridors and/or a long diameter) will become the bottleneck and again, the actual algorithm used, as long as it is reasonable, would not have much of an impact. However, in middle-sized systems, such as the ones with $R = 10$ run on our test areas, the particular algorithm we use can make a substantial difference in terms of the performance of the system.

Overall, the data speaks unanimously that the system designed in this chapter scales well both with the size of the area and with the number of robots.

Chapter 4

Hardware Implementation

In the previous chapter we presented a design of an efficient, fault-tolerant and scalable system for solving Distributed Area Search, under a set of assumptions for the hardware and the environment. In order to verify the practical applicability of this system, however, we need to verify that the set of assumptions is realistic and that it is feasible to build a system that meets them. This chapter presents our implementation of a prototype system, consisting of 12 low-cost robots, which can cover areas of size on the order of twenty square meters.

The low cost of the individual components is a requirement for every distributed system. Thus, we decided to limit our budget to \$4000 (or about \$300 per robot) because this is approximately the cost of a single robot, which is reliable enough to be capable of solving Area Search alone by itself[1].

We were unable to find any robot capable of reliably measuring its coordinates (relative to anything) in the \$300 price range. Therefore, we decided to build a system integrating two different off-the-shelf devices: one to serve as the navigation platform, and the other to serve as localization and communication platform.

[1]With the help of the localization system, of course.

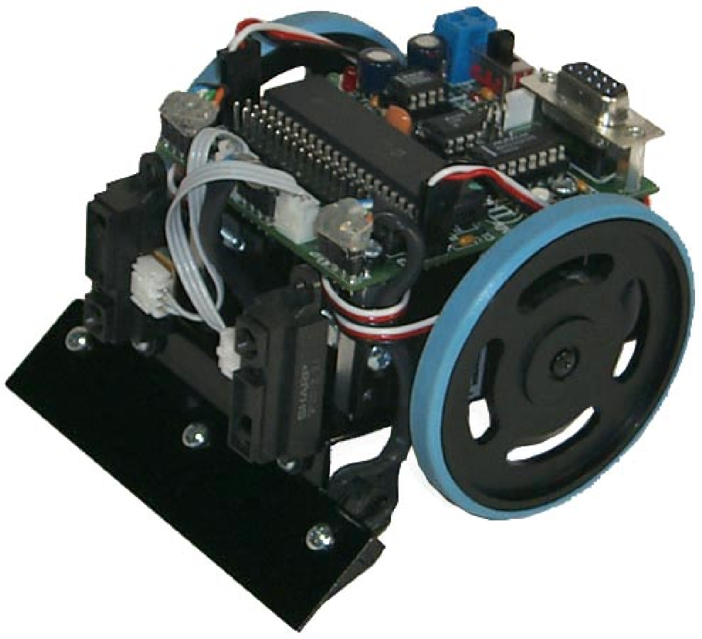

Figure 4-1: An individual off-the-shelf Mark III robot.

4.1 Mark III Robots

We selected the OOPic version of the Mark III robot [13] as the mobile platform for our system. Each Mark III robot is 10 cm long in all three dimensions, costs less than $100, and provides:

- A programmable OOPic microcontroller.
- A pair of independent motor-driven wheels, which can be commanded by the microcontroller.
- A pair of frontal IR distance measurement sensors with an operational range of 10 to 80 centimeters.
- A serial port, allowing the microcontroller to communicate reliably with another device.
- Several other features, which we did not make any use of.

Due to the numerous constraints on the microcontroller (mostly memory constraints) we decided not to implement the CCA logic on it. Instead, we use the Mark

III robot exclusively as a navigation platform. The program we implemented on the Mark III microcontroller uses the above-mentioned hardware to provide the following interface on the serial port:

- The robot can be commanded to go forward, go backward, turn, or stop. The first three commands always come with parameters specifying the velocity and the duration of the motion.

- When the robot completes one of the above operations, it sends a specific message over the serial port.

- When one of the IR sensors detects that the robot is 10 cm away from an obstacle, the robot stops and sends a message notifying the device on the other end of the serial port.

The Mark III robots generally manage to execute the given commands well. Common problems include:

- If the robot runs out of battery, it may stop responding to commands. This is rarely a problem on a single robot, but on a distributed system this problem appears on at least one of the robots often enough to be recognized as a frequent problem.

- The motors of the wheels are not very precise, so often times the robot will end up a few centimeters away from its desired destination, or a few degrees off of its desired angle of orientation.

- In some cases the IR sensors might not detect the nearby obstacle and then the robot might get blocked by the obstacle. This is often the case when a robot is moving nearby an obstacle, without facing it, and then the robot turns towards that obstacle, without having a 10 cm frontal distance to it at any point in time.

4.2 Crickets

The second device used in our robotic system is a Cricket node [3], serving both as communication and localization system, as well as as the processor running our algorithm. One Cricket costs about $200, thus fitting into our budget, and provides the following features:

- A programmable microcontroller capable of controlling the various subsystems of the device. The microcontroller is running TinyOS [24] and is programmable in nesC [6].

- A communication radio system meeting the assumptions of section 3.1 (i.e., best-effort delivery of messages and reliable correctness of delivered messages). The communication system of a Cricket broadcasts every message and does not expect acknowledgements back, so it is capable of working equally well as the number of recipients scales.

- A system for measuring the distance between any two Crickets facing one another (within some wide angle). Every Cricket is equipped with an ultrasound transmitter and receiver pair. The distance-measurement system works by having one of the Crickets emit a radio signal and an ultrasound signal at the same time, and then having the other Cricket measure the difference between the time of arrival of the two signals. Since the two signals travel with different speeds, the length of this delay is proportional to the distance between the Crickets.

- A serial port, allowing the microcontroller to communicate with another device, such as a Mark III robot.

The Crickets are generally much more reliable than the Mark III robots. Their microcontrollers fail rarely (almost always because of depleted batteries, which happens very rarely, as the Crickets are designed for low energy consumption). The communication system, as mentioned above, is a best-effort system, which is all that we require in section 3.1. The most problematic part of the Crickets is their distance-measurement system. There are two kinds of issues with it:

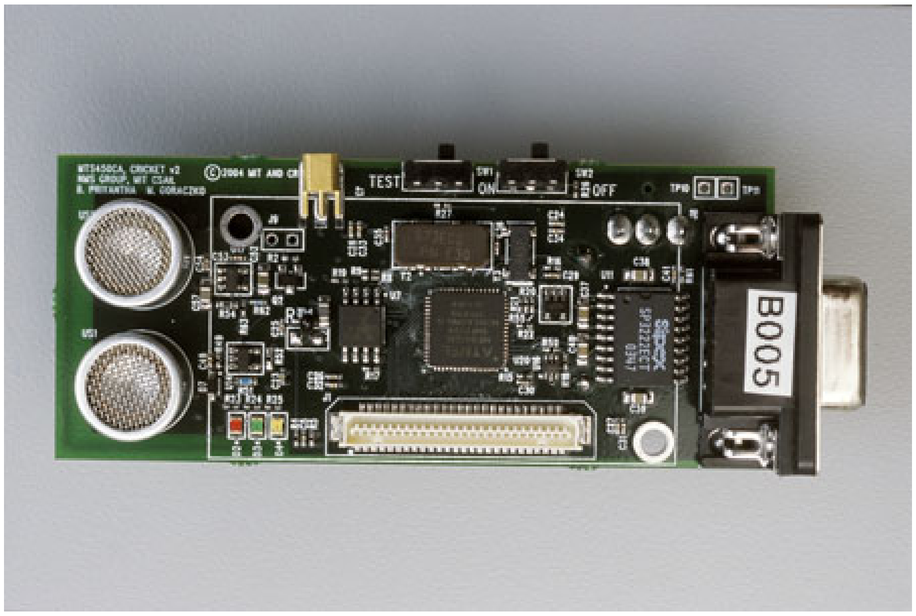

Figure 4-2: An individual off-the-shelf Cricket.

- If the angle between the directions of the ultrasound transmitter of the emitting Cricket and the ultrasound receiver of the receiving Cricket is above 60^o, the ultrasound signal rarely (if ever) manages to get to its destination. Thus, if two Crickets are positioned in such a way, we might get consistent, repeated failures of the distance-measurement system.

- The distance measurements are sometimes wrong. The most frequent scenario is if the direct line of sight between the two Crickets is interrupted. In this case the ultrasound signal arrives much later and this results in overestimates of the distance. Sometimes we can also observe underestimates, but they are generally much less frequent.

4.3 Localization System

In order to be able to prototype our location-aware system on our team of robot-Cricket pairs, we needed to construct a prototype of the universal localization system assumed in section 3.1. In order to do that, we employed the following setup.

Three Crickets, A, B and C, called "stationary" Crickets serve as the global frame of reference for all other Crickets. Each one of the three is positioned on the top of a separate 1.5 meters high tripod. A's tripod is positioned at the lower left corner of the area to be covered, with A facing the area. B's tripod is positioned at the lower right corner of the area, and C is located at the center of the uppermost boundary. All Crickets are oriented so that their ultrasound components are facing (within the above-mentioned 60 degrees) both the other stationary Crickets and any Crickets that would be located on the area ground[2], facing upward.

Cricket A is defined to have coordinates $(0, 0, 0)$. All that A does is periodically emit radio and ultrasound signals (so that other robots can measure their distances to A), attaching to every radio packet a monotonically increasing ID number.

Cricket B is defined to lie on the positive X axis, hence its coordinates are $(X_B, 0, 0)$. It learns the value of X_B by listening to A's signals and waiting for a long enough sequence of consecutive equal distance measurements. When B decides on a value for X_B, it starts emitting radio and ultrasound signals every time it receives such signals from A. B attaches its coordinates to its radio signals and it also attaches the same ID number it receives in A's signals (so that the mobile Crickets can pair the two signals coming from A and B with the same ID number).

Cricket C is very similar to B. The only difference is that it is defined to have coordinates $(X_C, Y_C, 0)$ with Y_C being positive. Thus, it waits for consecutive equal signals from both A and B in order to determine its coordinates. Also, after it decides on X_C and Y_C, Cricket C "re-emits" B's signals in the very same fashion as B "re-emits" A's signals. Thus, the mobile Crickets in the area periodically receive triplets of distance measurements from A, B and C. The ID numbers of these triplets can also potentially serve as a global clock in the distributed system.

Now that we have three stationary Crickets with known and reliable coordinates, each mobile Cricket D receiving a triplet of distances to A, B and C can compute its own coordinates. This is essentially solving a system of three equations and three variables (i.e, the three coordinates of D). This is the well-known problem of trilateration, which has a simple solution in three closed-form formulas for the coordinates.

[2]In fact, 10 centimeters above the ground.

The only uncertainty in the solution is the sign of the Z coordinate. However, since A, B and C are about 1.5 meters above the ground, and all moving Crickets are only about 10 centimeters above the ground, we know that the Z coordinate of D has to be negative.

The setup described above definitely provides a localization system, however the reliability of this system is very questionable. As mentioned in the previous section, distance measurements are unreliable. Even though the X_B, X_C and Y_C coordinates are reliable (because they are confirmed only after numerous consistent readings), the individual measurements between D and the stationary Crickets can be erroneous. Hence, we need to wrap the readings of this unreliable localization system with a filter that makes sure only confirmed, sensible readings are passed on to the client of the localization system.

In order to implement this feature, we assume that the area is flat. Thus, all coordinate readings have to be in the same plane (with Z coordinates approximately equal to minus 1.5 meters). Having this assumption in place, the localization system filter throws away every reading that is not within some margin of the above-mentioned plane. Given that the failures of the distance-measurement system are almost always overestimates, it is very hard for erroneous readings to end up in the "correct" plane, but it is nevertheless possible. Thus, the filter also requires each measurement to be repeated twice (within some margin) in order to pass it to the client of the localization system. This requirement practically filters out all remaining outliers, and during the tests of our system we have never observed an erroneous reading (above some reasonable margin) getting passed onto the command algorithm.

With this, we can conclude that the Crickets in our prototype system can indeed rely on a global (for the given area) localization system that gives them reliable coordinate measurements in some unbounded time, which nevertheless has some finite expectation.

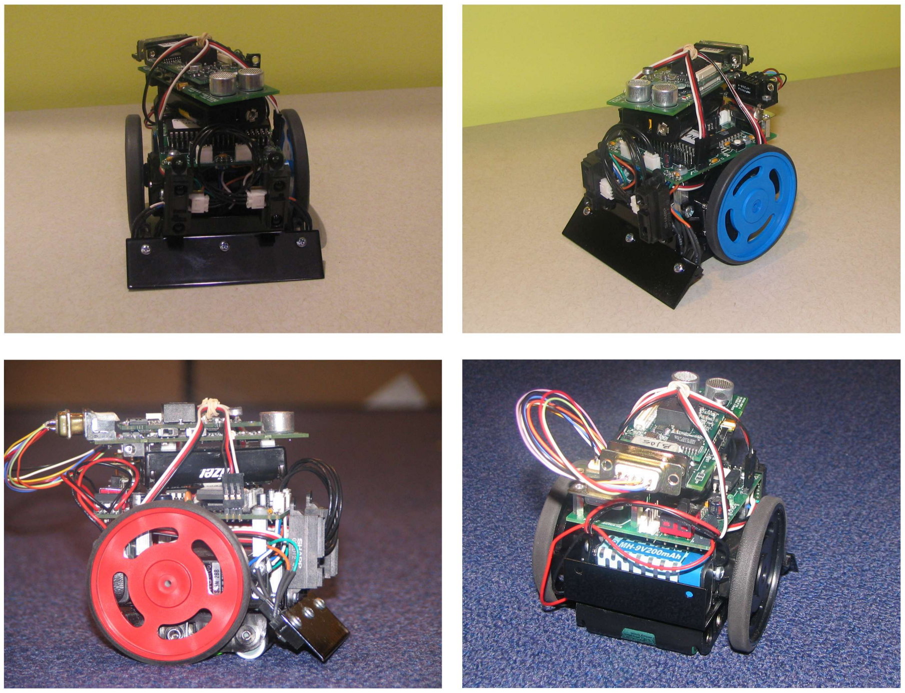

Figure 4-3: A coupled pair of a Cricket and a Mark III robot.

4.4 System Integration

Having this hardware and this infrastructure in place, we integrated the various parts into a system implementing the Compact Coverage Algorithm. This included the following steps (applied to each unit of the distributed system):

- The Crickets and the robots were connected via a cable through their serial ports, as shown on figure 4-3. As mentioned above, the Crickets are executing the algorithm, while the Mark III robots are taking navigational commands and responding back with a message indicating how the motion ended (either successfully, or because of detection of an obstacle in front).

- We implemented the feedback-driven navigation system assumed in section 3.1. The Crickets estimate the angle of orientation of their Mark III robot by looking at the sequence of commands and coordinate measurements. To a first approximation, a good and very simple method is to just take the difference between the last two coordinate readings[3] and then to adjust for any rotation commands

[3]In fact, take the last two readings, which were at least a few centimeters away from one another.

the Cricket has issued. A better method, although slightly more complicated one, is to use an Extended Kalman Filter [26]. Our system was based on the first approach (mostly because of memory limitations) and performed well enough for the purposes of the system. Most of the time, the robots were able to reach their destinations without ever having to turn around and go back. We had a few cases of oscillatory instability in the feedback controller, but they were rare and usually did not last forever. This, of course, is not a problem for our model, as the time taken by the navigation system is unbounded and so we can afford to wait for the feedback controller to stabilize. In the rare event that it never stabilizes, this is considered a failure and our algorithm is designed to keep working seamlessly with unexpected[4] robot failures.

- The IR sensors served as the basis for obstacle detection. However, as mentioned in section 4.1, the IR sensors sometimes fail to detect an obstacle and the robot gets blocked by the obstacle. Because of this, we implemented a software collision detector in the Crickets. The Cricket code would detect if the coordinate readings are consistently giving the same location, while the robot keeps reporting that the "move forward" command has been completed. Whenever the Crickets detect this situation, they command the robot to back off and then process the event as if the IR sensors detected the obstacle.

- Finally, sometimes the robots would run into one another (either detected by the IR sensors, or by the software collision detector). In order to prevent reporting obstacles where there are none, we had to implement a subsystem that, whenever the robot collides, detects if the object of the collision is an obstacle or another robot. In order to enable this, every robot repeatedly transmits its most recent coordinates, as part of its state variable[5]. Then whenever a collision is detected, the Cricket which detected the collision checks the coordinates of all other known robots. If one of them is directly in front, then the collision would be detected as a robot collision and no obstacle would be recorded in this

[4]In fact, this term is inappropriate, as every robot failure is expected in our system.
[5]See section 3.4.1.

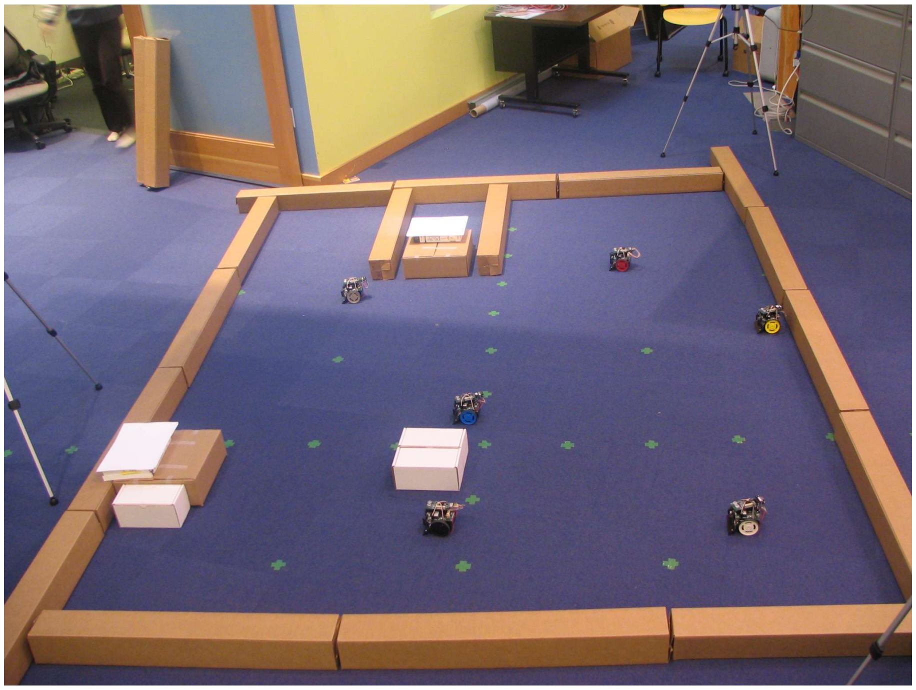

Figure 4-4: A test maze with 6 robots.

location. Otherwise, the map of the robot would be updated with an obstacle at its present cell.

The system outlined above addresses all problems of the Mark III robots and the Crickets, as listed in the above two sections, reducing them to things that our location-aware system design copes successfully with (such as long delays and stopping failures). This makes our prototype system satisfy the assumptions of the Compact Coverage Algorithm, hence we can proceed with running and evaluating the prototype in practice.

4.5 Results and Evaluation

We ran a number of tests on the prototype system in order to evaluate its performance and to assess whether the assumptions in section 3.1 are in fact realistic. The tests were run on mazes constructed of cardboard obstacles, as photographed in figures 4-4 and 4-5.

The results of the test runs matched our expectations. Even though one or more

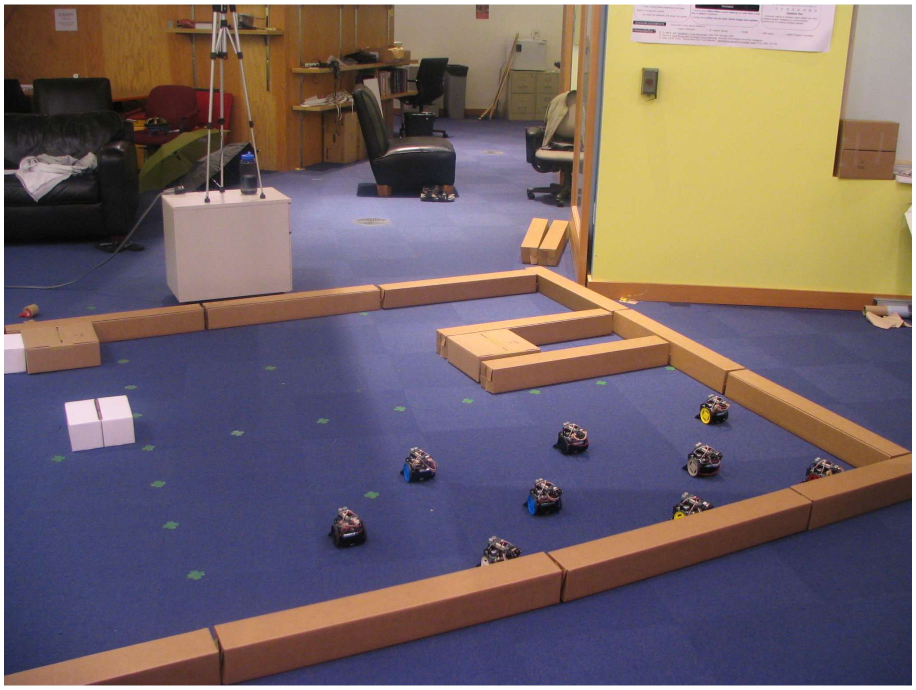

Figure 4-5: Part of a test maze with 9 robots..

robot would get "stuck" multiple times (mostly because its Cricket has tilted slightly and so it becomes hard for it to receive the ultrasound signals from the stationary Crickets), the system as a whole would function well all of the time. This means that we have achieved our goal of creating an efficient and fault-tolerant system. The efficiency of our algorithm was demonstrated mostly by the simulation runs in section 3.6. The small scale of our prototype prevents us from doing a rigorous study of the efficiency of the real-world system. Nevertheless, the log files we collected from the robots (and the video, which we used to verify the correctness of the log files) show that the robots wasted very little time due to communication problems or bad planning. Table 4.1 shows statistics of a test run with 9 robots on the photographed maze (12 by 10 cells, with a cell side length of 30 centimeters). The numbers in each cell are the ID numbers of the robots that thought they discovered that cell first. Clearly, the instances in which miscommunication caused two or more robots to repeat one another's work are very few. The cells which have no numbers are all unreachable obstacles.

Based on our experience with the prototype system, we can conclude that the

	4			6	6	5,6	6	6	1	1	
4	4	4	6	6	5,6	1	1	1	1,4	1	1
4	4	6	6	6	5	1	7	7	7	7	7
4	4	4	4	4	5	1	7	7	7	7	
4	4	4	4	4	4	4	4	7	5	8	
4	4	4	6	6	6	6	4	7	5	5,8	8
6	6	6	6	6	6	6	4	7	5	8	5
6	6	6	6	3	3	1	4	1	5	2	2
3	3	6	6	3	3	9	2	1	2	2	5
	3	3	6	3	5	9	2	2	2	5	

Table 4.1: Pattern of discoveries in the real-world prototype system.

assumptions in section 3.1 are indeed realistic and that implementing a real-world system meeting these assumptions is feasible and does not pose any hard challenges.

Chapter 5

Autonomous System

This chapter presents a different solution to the Distributed Area Search problem. Unlike in chapter 3, here we do not assume the existence of any global localization system. In this scenario, the robots have to solve Area Search alone by themselves, without the help of any external system, agent or infrastructure.

This variation of the problem is much more difficult than the one addressed in chapter 3. The goal of finding a solution is definitely much more ambitious in this scenario and, to the best of our knowledge, the work presented here is a first attempt at solving this problem. That is why we relax many requirements that we imposed on the location-aware system. For example, in this chapter we do not aim for high efficiency or fault-tolerance. The only goal of this system is to solve Area Search correctly, that is, to have the robots search through the whole given area and accomplish the task successfully in finite amount of time.

We claim that this goal is difficult enough for the following reasons. Imagine a single robot trying to accomplish Area Search without the help of an external localization system. First, this robot has to be equipped with expensive tools for sensing the environment, as otherwise the odometry errors would cause the robot to lose track of its current location and its history of visited locations. Second, even with the expensive sensor equipment the robot would still be likely to lose its global coordinates. The reason is that every sensor, no matter how accurate and expensive, has some error in measuring distances and angles. Since the robot has no access to any global coordinates or any stationary object (with known coordinates), the robot can only measure distances and angles relative to its previous state. Since these new measurements also have some error, our new state has an error, which is a combination of single-measurement errors.

If we think of a measurement as a random variable with variance σ, then after making N steps and measurements, probability theory and the Law of Large Numbers tell us that our current estimate of the state variables (such as location and orientation) will have a variance on the order of $\sqrt{N} \times \sigma$. Thus, if the system keeps working for long enough, it will inevitably accumulate so much error that it would be unable to distinguish between two similarly looking, but substantially different scenarios. For example, if the robot walks in a circular corridor with a very large radius, it will be practically impossible for this robot to distinguish between this case and a case in which the corridor is a large outwardly expanding spiral.

The only way the robot can possibly distinguish between these two cases is if the robot uses some external mechanism to interact with the environment, for example, if the robot can place uniquely identifiable marks on the ground and rely on those marks remaining there until the end of the operation. Even if we allow this, however, distinguishing the marks from a distance might be very hard, unless the marks are active transmitters of some unique signal, in which case we have essentially built a distributed system, not a single-robot one.

The above analysis is equally valid in a case with a small, limited number of robots. If this small team is put in a large enough environment, they will either have to split ways (i.e., lose one another's relative coordinates) or move away from the origin, in which case they would suffer from the same error accumulation described above. Thus, solving Area Search in a non-distributed manner (and without the use of a multitude of active helper agents, or some other external way of getting limited-error measurements on a global scale) is practically impossible, at least for a set of worst-case environments.

In this chapter we present an algorithm, which uses a distributed robotic system to solve Area Search without external help. The number of robots required for our system depends on the complexity of the environment, as described in section 5.5 below.

5.1 Model Assumptions

The assumptions under which we design this system are related to the assumptions of the location-aware system[1] in many ways, but there are several important differences.

First and foremost, this system is not designed for seamless fault-tolerance. We assume robots are not going to fail, and if they fail, we assume that we can detect this event and replace the failed robot with another one. Section 5.7.1 below discusses these issues in more detail. For now, we just assume we should not worry about robots failing.

The second important difference is the control flow of the system. The location-aware system was concerned with efficiency and thus in that system all robots worked simultaneously in parallel. Under those goals, it was definitely important to have all robots moving all the time, which naturally required every robot to take care of itself and to make decisions for itself. This is not the case under the model in this chapter. Efficiency is not a concern in our case, but accurate execution of the algorithm definitely is. Thus, under the assumptions of this chapter, for the purpose of simplifying our proofs and analysis, we can afford to have a single point of decision making, and also to move the robots one at a time and not in parallel.

Since we assume no failures and we assume the same unbounded-in-time, but reliable communication system we assumed for the location-aware system[2], we can safely assume that a single robot commands the whole group. The decisions of that single robot are communicated to the appropriate robots, which in turn send acknowledgements of task completion (and probably other information) back. Unacknowledged

[1]See section 3.1.

[2]One might argue that if we assume such a communication model, it is not an autonomous system anymore. However, our view of the communication system is that sending a message that reaches far enough is an action of the robot, not of the environment: it is the robot that has to actively "push" a strong-enough signal through the radio waves, not the radio waves acting in assistance of the robot.

messages are resent until either an acknowledgement is received, or the central algorithm decides that the robot has failed. The failure detectors and the event of failure of the decision-making robot are discussed in more detail in section 5.7.1. For the rest of the chapter we will work under the assumption of a reliable communication system and a reliable single point of control.

Another model difference with the location-aware system is that in the autonomous system we do not have the global localization subsystem from section 3.1. In its place we assume that any two robots are able to measure, within some small error ϵ, the linear distance between themselves, provided that there is a direct line of sight between them. Having this feature, any three robots A, B and C, which have direct line of sight between each other, can setup a local coordinate system much like the one described in section 4.3. Then they can provide a localization system to all robots, which have a direct line of sight with A, B and C. This localization system will, of course, only give coordinates that are relative to the positions of A, B and C.

Having these local coordinate systems and their corresponding relative localization systems, we can assume that our robots have the same localization and navigation systems as the ones in section 3.1, but with respect to those local systems. That is, we assume that every robot can get reliable positioning information, relative to the positions of three other robots, and it can also navigate reliably over these coordinates.

Our assumptions with respect to the environment of the robots (i.e., free space, obstacles, etc.) are very different from the ones in section 3.1. Here we do not introduce a discrete grid world. However, we make two other strong assumptions. First, every point in the world is either a Free point, which means that a robot can both walk over it and see visually through it, or the point is an Obstacle point, in which case a robot can neither walk over, nor see through the point. This ties the visibility in the area with the "passability" over the points in the area, which is a strong assumption, but nevertheless reasonable and correct in many scenarios.[3]

The second strong assumption we make with respect to the surrounding environment is that the borders and obstacles of the world can be represented as simple

[3]This assumption was first introduced in [19].

(i.e., non-self-intersecting) polygons. This means that any curvature can essentially be represented as a sequence of many sharp, linear corners in a way that is both coarse enough to be easy to process by the robots, and fine enough to preserve the correctness of the area coverage. For the rest of this chapter the sides of the polygons are called “walls” and the vertices of the polygons are called “vertices”.

In addition to the above assumptions, we also assume that every robot has the following reliable abilities:

- Every robot, which is positioned right next to a wall, can follow that wall in a desired direction.

- Every robot that is following a wall can detect a vertex, which marks the end of that wall. This holds for both concave and convex vertices.

- Every robot A, which is moving in some direction, can immediately detect if the direct line of sight between A and some other robot B is interrupted by an obstacle. Also, A can immediately detect if a direct line of sight between A and another robot C appears (that is, if it was previously interrupted by an obstacle).

Finally, one last assumption for the environment is that it is stationary (i.e., obstacles remain exactly where they are and no new obstacles appear). This also means that we assume that the robots' readings of the environment are consistent. For example, if there is a small tunnel in a wall, right about the size of a robot, the robots that pass by this tunnel should always decide consistently if this is a branch that should be taken, or if this should be ignored and treated like a part of the wall (because a robot cannot pass through it). This assumption is more thoroughly discussed in section 5.7.2 below.

5.2 Environment Representation

Since all boundaries and obstacles are polygons, the free space that is to be covered is almost also a polygon (the only difference is that it may have “holes” of polygon-

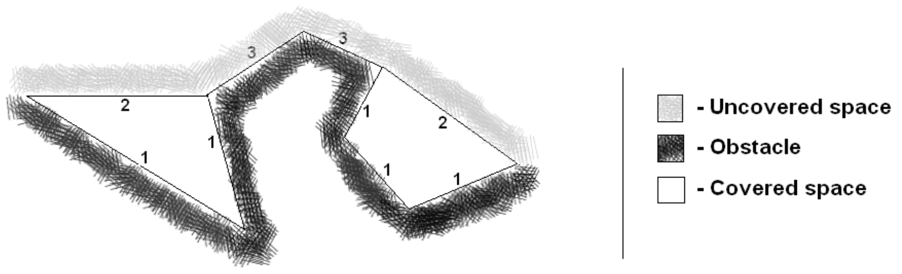

Figure 5-1: An illustration of the different types of C sides.

shaped obstacles inside). Hence, it seems natural to represent the covered area at any point in time with a polygon, and this is exactly what we do in our system. At any point in time, the area that has been covered, plus the obstacles completely surrounded by such area, is represented by a polygon C.

The robots do not know what is outside C, so except for C, all that is known to the robots is the types of all sides of C. There are three types of such sides:

1. Sides which are between covered area and obstacles: These sides are the least interesting ones, as they are already covered, and they have no relation to the future.

2. Sides between covered area and uncovered free area: These are essentially the "front lines" of our algorithm.

3. Sides between obstacles and uncovered areas: These are "passive" segments that essentially serve as connectors between the front lines, and that will eventually be "swallowed" by the front lines.

Figure 5-1 gives an example of the representation of a given environment with labeled types of the sides of C.

Another important thing to keep in mind is that we do not have a single global-coordinates localization system, so we cannot have C in memory in global coordinates. We can only represent the vertices in or on the border of C by local coordinates, relative to nearby vertices. That is why we represent C by a covering set of disjoint triangles (a.k.a. triangulation of the polygon). For every triangle we store the relative coordinates of every vertex with respect to the other two vertices (i.e., in a right-handed coordinate system where the first of the other vertices is defined as the origin, and the second is defined to lie on the positive X-axis).

We also store all of the nodes that are in or on the border of C as the nodes of an adjacency graph G. Two vertices are connected in G if and only if they either share a triangle, or they are the vertices of a type 3 segment (i.e., one which does not participate in any triangle, and which serves only as a "connector" of disconnected triangle-filled regions of C).

Having this in place, we enable groups of robots to travel anywhere (i.e., from any vertex to any other vertex) within C. The way that they accomplish this is by performing a series of some of the following procedures for moving to an adjacent vertex in G:

- If a robot is to move from a vertex A to another vertex B and A and B are the two ends of a wall, then the robot can get from A to B by directly applying its wall-following primitive and its vertex-detection primitive.

- Since every vertex shares a wall with at least one other vertex, we can do the following. If robots A, B and C are on vertex X, then A can stay on X, B can follow X's wall to an adjacent vertex Y, and C can use the coordinate system spanned by A at X and B at Y to go to the third vertex of a triangle shared by X and Y.

- Once three robots from a group are at the three vertices of a triangle, the group can go to an adjacent triangle (i.e., one that shares a wall with the first) by having a fourth robot go to the new vertex (i.e., the one in the new triangle, which is not in the old triangle). In order to do that, the fourth robot will use the coordinate system set up by the vertices on the shared edge, and the known relative coordinates (in that same coordinate system) of the target vertex.

- If the robots spanning a triangle would like to gather at one of the three vertices (e.g. to continue on a type 3 side of C, starting at that vertex), they can easily do that by navigating towards the robot at the desired vertex, using their pairwise distance measurement capabilities in a feedback loop. Since the robots are in a triangle with no obstacles in it, following the gradient of the distance measurements should not be a problem.

Having these procedures, the robots can go from any vertex in C to any of its adjacent (in G) vertices. Since G is a connected graph, which includes all vertices in C, this means that the robots can travel from anywhere to anywhere within C.

5.3 DFS Exploration Algorithm

Now that we have a good environment representation, which allows any group of robots to travel anywhere any other group has ever been, we can proceed with presenting an algorithm for searching through the area, with the safe assumption that any vertex in C is reachable for every robot.

In a fashion similar to the location-aware system, we will also assume that the robots start together, right next to a wall on the boundary of the area. To make things clearer and more strictly defined, we assume that initially C includes exactly one triangle, which has two type 1 sides, both part of the area boundary, and one type 2 side. All robots are assumed to start inside that triangle, with three robots at its three vertices (all of which are vertices on the boundary of the area).

Having an initial C, we only need to provide an algorithm, which given some C finds a way to expand C and add an additional triangle to it. If we have such an algorithm, we can apply it again and again until the whole area has been covered (i.e., until C includes the whole area).

In this section, we propose the DFS Exploration Algorithm for expanding C. The name comes from the way C expands when we run the algorithm on a network of long corridors. Essentially, we observe a depth first search on the area, with a pair of robots remaining at every branching point, in order to detect loop closures. Figures 5-2 and 5-3 illustrate how the algorithm behaves in the above-mentioned corridor network case, as well as in a case with more wide-open areas. In both figures the

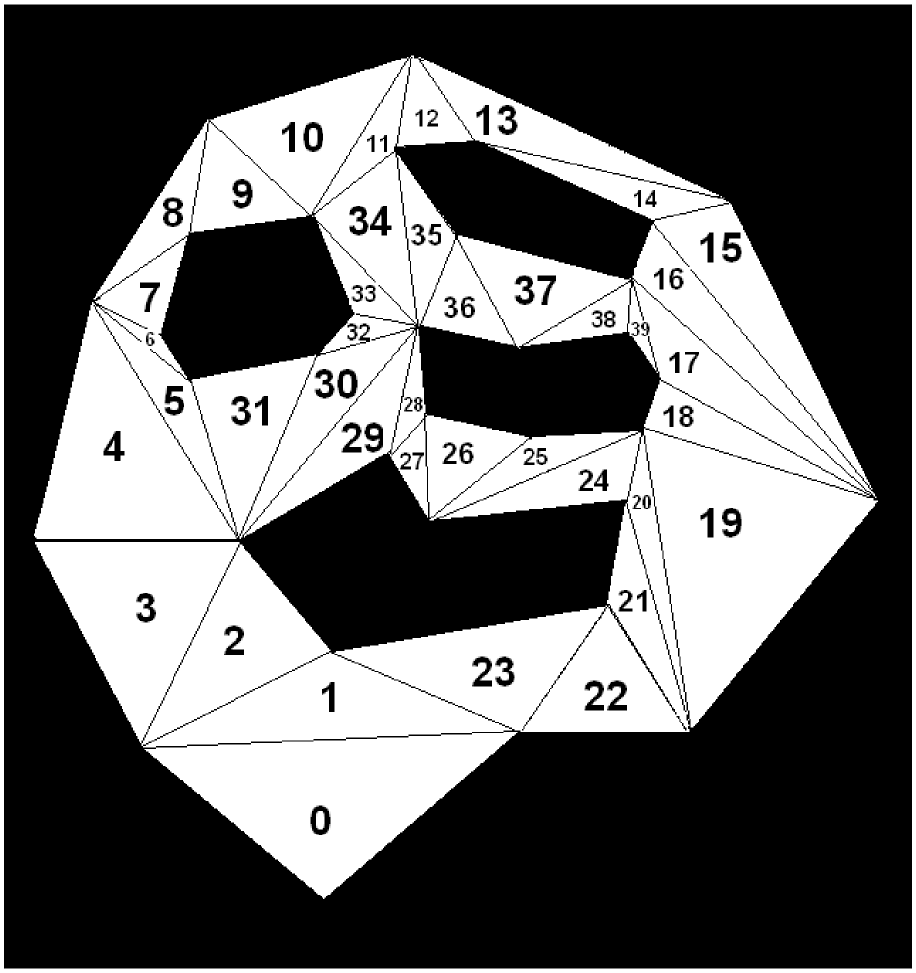

Figure 5-2: The order in which the DFS Exploration Algorithm covers triangles in a corridor network.

numbers indicate the order in which the triangles were covered (i.e., the order in which they were added to C).

As discussed in section 2.2.2, the DFS Exploration Algorithm is built over an algorithm described in [19], and it is defined in the following steps[4]:

1. Define a variable F, which points to one of the sides of C. Initialize this variable to the only non-wall side of the initial C. The F variable essentially points to the side of C, which we will try to expand in the next steps.

2. Define as an invariant over the algorithm that there will be two robots at the

[4]At the end of this section, the triangles in figure 5-3 are related to their corresponding steps in the algorithm.

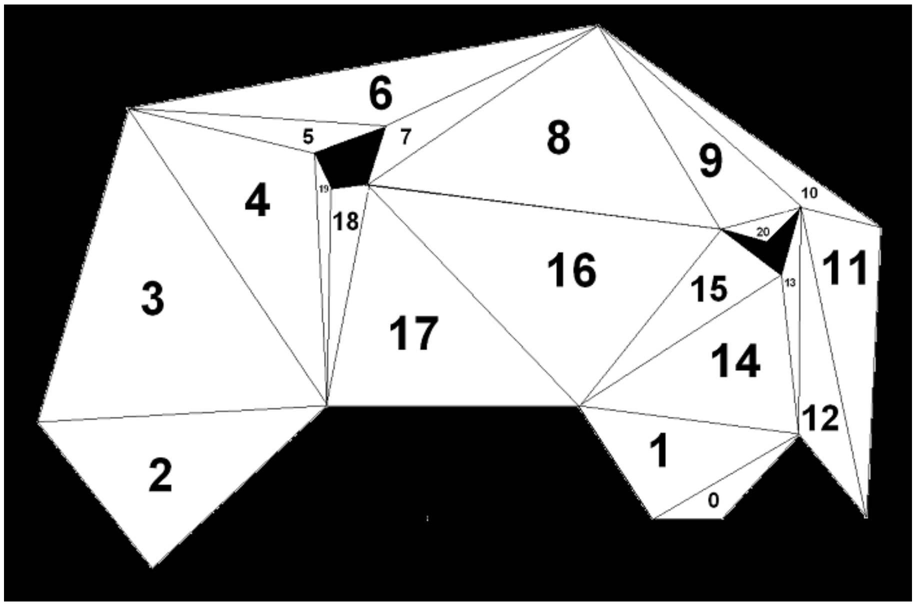

Figure 5-3: The order in which the DFS Exploration Algorithm covers triangles in a more wide-open case.

ends of any side of C, which is of type 2 (i.e., a front line between covered and uncovered free space). Thus, every time the robots expand C, adding a new side of this type, two robots shall be commissioned to the two ends of that new segment. Initially, the only such segment is the initial F. Similarly, whenever a triangle gets added to C and one such segment is internalized (i.e., the latter is not a side/border of C anymore), the two robots at the segment's ends should be released and reused for other purposes.

3. If F is facing a wall (i.e., if it is a type 1 segment), or an area that has already been searched through by the robots, then backtrack the history of covered triangles, set F to the first type 2 (i.e., front line) side of S, and repeat step 3. If there are no such sides left, terminate the algorithm.

4. At this point, the F segment is certainly of type 2 or type 3, which means that there is unexplored free space on one of its sides. We can measure the angle of that space at both vertices of F.[5] The way we accomplish this is by simply

[5]Imagine a very small circle around a vertex. The arc length (in degrees) of the portion of the circle, which is part of the unexplored free space, gives us the "angle" of that space at the vertex.

measuring the angles of the other two types of space (obstacle and covered free space). We know the angle of the covered space (as we have the relative coordinates of all vertices in C) and we can measure the angle of the obstacle section of a vertex by using the wall-following primitives and the coordinate system set up by the robots at the ends of F.

Clearly, if the angle of the uncovered space is more than 180^o, we cannot use a third robot to walk along the boundary of that uncovered space, as it will lose line of sight with the robot at the other end of F. Hence, in this step we do the following. If the uncovered space angle is less than 180^o around at least one of the vertices of F, then proceed to step 5. Otherwise, go to step 8.

5. Send a new robot, denoted by Z, along the edge of the uncovered free space, starting at the vertex at the end of F, which has the smaller angle. If this edge is of type 2 (i.e., the uncovered space borders with covered free space), then by the invariant defined in step 2 above, there should be two robots at the ends of this edge, hence Z can use their coordinate system to navigate along the edge. If the edge is of type 3 (i.e., the uncovered space borders with an obstacle), then Z can use its wall-following primitive to follow the edge. In both cases, Z constantly maintains line of sight (and thus distance measurements as well) with the two robots at the ends of F.

   Again, in both cases, if Z reached the vertex at the end of the edge while still keeping contact with the robots on F, then proceed to step 6. Otherwise, if the line of sight between Z and the robot at the opposite end of F was interrupted by an obstacle, go to step 7.

6. Add the triangle between the endpoints of F and the vertex of Z to C.[6] The newly formed side of C (the line between Z and that end of F, which Z did not start from) is of type 2. Set F to that new side and make sure to keep the invariant defined in step 2 above. Go to step 3.

[6]See the paragraph after the enumerated steps for a more detailed description.

7. Once Z has detected a line-of-sight interruption to a robot X, Z should go to the vertex that caused this interruption. How this is accomplished depends on the specific robotic platform used. One simple way to do it is to have the robot go back and forth between the areas with and without line of sight to X, while going towards X at the same time. When Z bumps into an obstacle, this is the vertex that Z has to go to. Once this is achieved, the steps are as follows.

   Add the triangle between the endpoints of F and the new vertex of Z to C. The two newly formed sides of C (the lines between Z and the ends of F) are of type 2. Set F to the left one of these two sides (assuming forward is defined as the direction of Z as observed from F) and make sure to keep the invariant defined in step 2 above. This would require taking two additional robots, as the number of type 2 C sides is increasing by one. Go to step 3.

8. Have a new robot go to the left end of F (assuming forward is defined as the direction of the uncovered area). Have that robot wall-follow the obstacle from there (on the uncovered side) until a vertex is reached. Add the just-walked segment to C as a type 3 side and set F to it. Go to step 3. The intuition behind this step is that if the robotic team has reached a place where the border is concave, it should continue following that border until it becomes convex and then start adding triangles to C from there. An illustration of this is the “jump” from triangle 1 to triangle 2 on figure 5-3.

It is important to note that when we say “add this triangle to C”, this does not simply refer to a data structure operation, but rather refers to the process of actually covering (i.e., searching through) the mentioned triangle and only after that adding it to the C data structure. Covering this triangle is always possible because every time we use the phrase, there are always three robots at the three vertices of the triangle, and the triangle is always free of obstacles. The three robots at the vertices can set up a coordinate system, and then the other robots can execute the Compact Coverage Algorithm described in chapter 3 to search through this triangle, taking the local coordinate system and the relative localization system as the “global” coordinate/localization systems in this subproblem.

The triangles on figure 5-3 might serve as a good illustration of the various cases, described in the above steps. The transition from triangle 1 to triangle 2 is of the case that goes through step 8. Triangles 4, 8 and 9 are constructed through step 7. It is important to note that this is not the case for triangles 14 and 16, as they are only illustrations of the case when a robot in step 5 walks along a type 2 edge (in the case of triangle 14, this edge is its border with triangle 1; in the case of triangle 16, the edge is the border of triangle 8). All other triangles in the picture are constructed through step 6 above, with the transition between triangle 19 and triangle 20 illustrating the backtracking described in step 3.

5.4 Proof of Correctness

The proof of correctness of the DFS Exploration Algorithm is as follows.

- The algorithm will never terminate without having covered the whole area first. In order for the algorithm to terminate, there must be no type 2 sides of C. If there are uncovered parts of the area, however, then C will definitely have type 2 sides, because the area is connected and thus, if we have both covered and uncovered area, there must be a border between the two. Hence, if there are uncovered parts, the algorithm cannot terminate. Hence, if the algorithm terminates, it means there were no uncovered regions in the area.

- The algorithm will certainly terminate. The number of triangles and type 3 segments we can add to C is finite, as these are limited by the number of vertices in the area, which are finite. The algorithm never stops passing through step 3 until termination. Hence if the algorithm does not terminate, it must have gone infinitely many times through step 3, which means it must have gotten there by not adding anything new to C infinitely many times (because the number of times it can add something to C is finite).

  The only way this can happen is if we go infinitely many times through step 8, which would be a case in which we endlessly iterate over vertices with more than 180^o sections of uncovered area. However, such a situation would be impossible, as it would mean that the uncovered area surrounds C. However, C starts at the border of the area, by definition, and the border clearly surrounds the free space (including the uncovered), by definition. Hence we reach a contradiction, and hence we reach the conclusion that step 3 cannot be executed infinitely many times. Therefore, the algorithm always terminates and by the above bullet, it always terminates correctly.

It should be pointed out that the lack of global coordinates does not influence the above logic. No robot is ever in danger of taking misinformed decisions, that is, a robot can never walk over covered area thinking it is uncovered. The reason is that every time there is a link between covered and uncovered area (the type 2 segments) the invariant postulated on step 2 assures us that there will be a pair of robots marking the boundaries of the covered area C.

5.5 Complexity Bound

We do not have infinitely many robots, so in order for our algorithm to be realistic, we need a bound on the number of robots needed. This bound will clearly depend on the particular shape of the area and the particular arrangement and shape of the obstacles. If we have a very complicated case with many obstacles and many branches of the free area, we will inevitably need many robots, as without the necessary number of robots, the system would be unable to detect loop closures, and thus, would be prone to either terminating with incomplete coverage, or never terminating.

In order to bound our algorithm, we define the following complexity measures:

- The complexity of a polygon is equal to one plus the number of concave angles of that polygon.

- The complexity of an obstacle or area boundary is equal to the complexity of the polygon representing that obstacle or boundary.

- The complexity Q of a given Area Search problem is equal to the sum of the complexities of all obstacles in the area and the boundary of the area.

An important observation is that if we split an obstacle at one of its concave angles, so that the resulting two obstacles have convex angles at this vertex, we preserve (or maybe even reduce) the complexity of the obstacle. Thus, we can think of any obstacle as a set of many convex obstacles, without incrementing Q. That is why from now on, we will think of all obstacles as being convex and being in number equal to Q minus the complexity of the border.

We simulated the algorithm on a number of test cases and in all of them the number of robots needed turned out to be very close to $2Q$. Thus, the requirements (in terms of number of robots) of our algorithm seems to be bounded by $O(Q)$. Here is an intuitive description why (i.e., an informal proof).

When the robots are sweeping between obstacles (i.e., looping through steps 3, 4, 5 and 6), we always have one robot wall-following one obstacle on the left, and another robot wall-following an obstacle on the right (with forward being the direction of progress).[7] The algorithm has to leave robots behind only when it deviates from this loop, and has to go through step 7 or step 8. Also, in all three cases (going through steps 6, 7 or 8), the robots always stick to the left border of the uncovered area. Thus, we have the robots circumnavigating the outermost layer of uncovered territory in a clockwise direction, leaving a pair of robots behind only when the obstacle to the right of their path is replaced by another obstacle (step 7), or when there is a concavity on the boundary (step 8). In addition, the algorithm also collects the robots left behind on the previous circumnavigation, as they are not facing uncovered area anymore. Hence, we are only interested in the number of step 7 and step 8 events[8] happening during one circumnavigation of the uncovered area (as this number is proportional to the maximum number of robots we will have to leave behind at any one point in time).

[7]Figure 5-2 might serve as a good illustration of this process.

[8]In fact, we are interested in step 8 events only when F is of type 2, as otherwise no robots are left behind.

Let us denote the number of obstacles[9] inside the uncovered area by K, the number of obstacles "swallowed" in C by L and the number of concave angles on the boundary of the area by M. Then we know $Q = K + L + M$.

The number of times the algorithm gets to step 8, with F of type 2, in one circumnavigation is clearly bounded by $L + M$. The event leading to step 8 (with F of type 2) can happen only once per "swallowed" obstacle, because the obstacles are all convex and so whenever this happens, it will be followed by a series of passes through step 8 with F of type 3, until the robots leave the obstacle behind. This is essentially the robots "sliding" down the obstacle, looking for a concavity, which will never come while they are on the convex obstacle. The degrees of the uncovered area will keep being greater than 180^o, until we depart from that obstacle.

This same event (step 8 with F of type 2) can happen on the border no more than M times. This is obvious, since the algorithm goes to step 8 only when there are uncovered areas with angles over 180^o at a given vertex. There are only M vertices that have at least 180^o in non-obstacle surrounding, hence this event cannot happen on the boundary more than M times. Thus, the total number of robots we have to leave behind, due to executions of step 8 is limited by $O(L + M)$.

In addition to the above, we claim that the number of times the algorithm executes step 7 in a circumnavigation is proportional to K, for the following reasons. We can sort the obstacles in the order in which they were first touched by a robot (in this circumnavigation). Each one of the (at most K) touched obstacles gets an R number denoting its rank in that sorted order (1 being the R number of the first touched obstacle, 2 being the R number of the second, etc). Since the obstacles do not cross one another, if an obstacle with a lower R intercepts the sweeping between the boundary and an obstacle with a higher R, this would mean that the latter has been fully surrounded by the robots, and so it will never be met again in this circumnavigation (i.e., it is impossible to touch obstacle A, then touch obstacle B, then A again and then B again).

Now, we can present the circumnavigation as a sequence of R values: the R values

[9]These are the convex obstacles in which we decomposed the original ones.

of the obstacles on the right side of the robots, sequenced in time. The algorithm is in step 7 (thus having to leave a pair of robots behind) only when the R value in the sequence changes, that is, if one obstacle intercepts the sweep between the boundary and another obstacle. Obstacles cannot intercept their own sweeps, because they are convex.

We know that the R values in the sequence can decrease only if an obstacle is never to be met again (because if the sequence decreases from X to Y, it means that the robots touched Y, then they wall-followed X, then they touched Y again, and as we know from above they cannot touch X again). Also, the R values in the sequence can increase only as many times as they decrease, because the sequence starts with $R = 1$, by definition, and ends with $R = 1$, because the circumnavigation always ends where it started. Thus, the total number of changes in the sequence is bounded by $2K$.

Overall, we showed that the number of step 8 executions (with F of type 2) is $O(L + M)$ and the number of step 7 executions is $O(K)$. Hence, the total number of robots we have to leave behind at any point in time is $O(Q)$. This means that if we are to solve Distributed Area Search with the DFS Exploration Algorithm on an area with complexity Q, we would need only $O(Q)$ robots to accomplish the task.

5.6 Hardware Implementation

Unfortunately, we did not have the resources to implement a full-scale prototype of our autonomous system. However, we were able to build a small, proof-of-concept mini-system, which demonstrated that the same cheap hardware we used in chapter 4 can in fact be used in a system like the one presented in this chapter.

There are three extra features assumed for the robots of the autonomous system, as compared to the robots in the location-aware system:

1. Ability to measure relative distances between one another.

2. Ability to detect the presence or absence of a direct line of sight between any

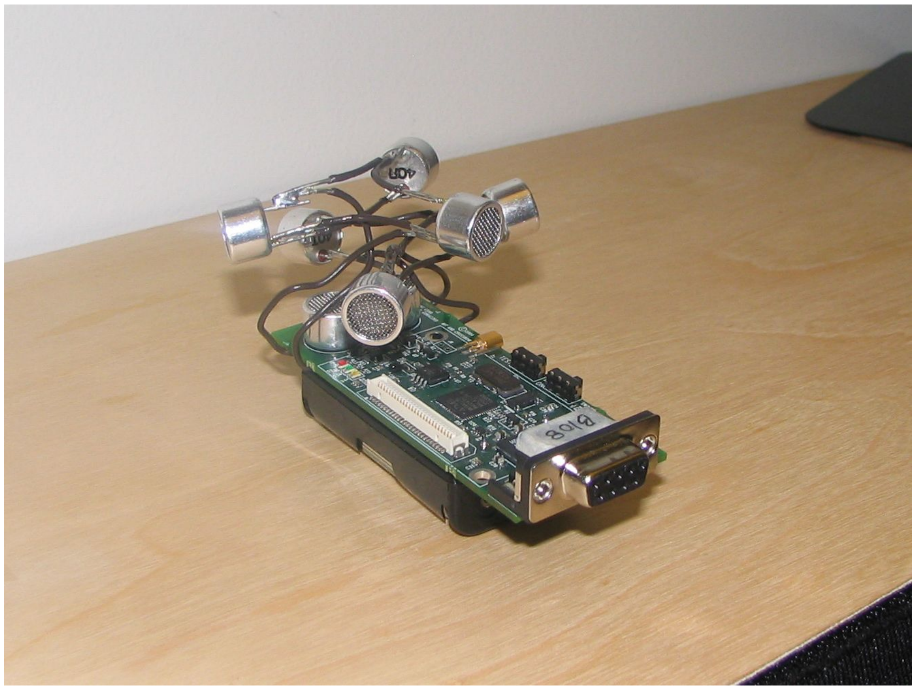

Figure 5-4: A modified Cricket.

two robots.

3. Ability to follow walls and detect corners.

In order to enable these features on our hardware, we had to make some modifications to our hardware units. The Crickets are designed specifically for distance measurement, but unfortunately they have a limited angular range, preventing them from being able to work in all 360 degrees. In order to make the Crickets "omni-directional", we had to modify each one of them, soldering three more ultrasound transmitters and three more ultrasound receivers to the terminals of the corresponding original components. The three transmitters are positioned on top of the Cricket, facing directions parallel to the ground and 120^o apart from one another. The three receivers are positioned on top of the three transmitters in a very similar fashion. Figure 5-4 is a photograph of a modified Cricket.

With the new equipment, every Cricket in our mini-system was able to measure the distance to any other Cricket, as long as the two had a direct line of sight. This precisely met the assumption of our model, as outlined in section 5.1 above.

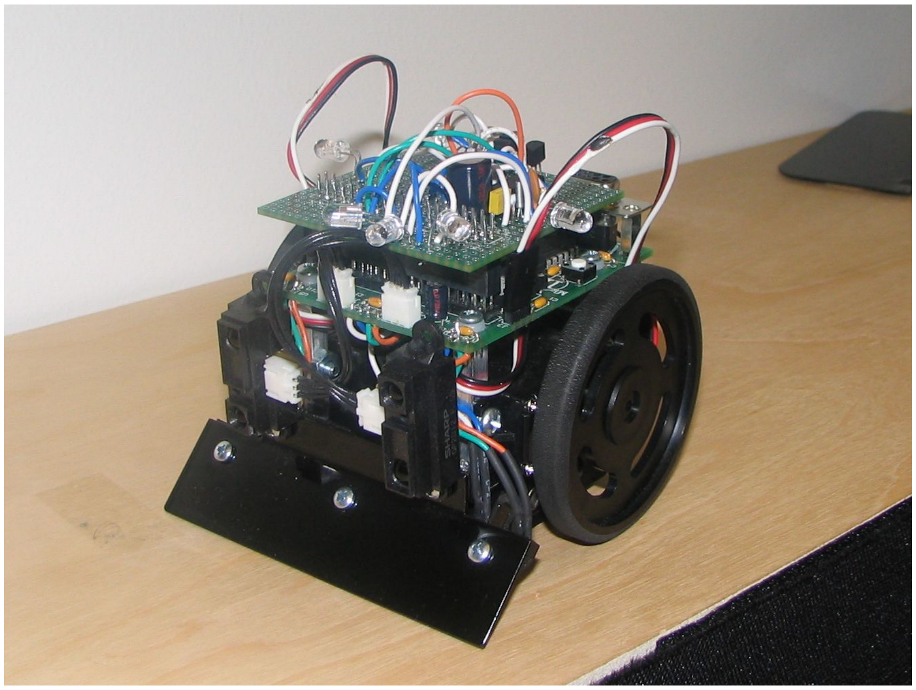
Figure 5-5: A modified Mark III robot.

Even though Crickets are affected by the presence or absence of a direct line of sight, they can hardly be used for the purpose of detecting whether such a line is present. Ultrasound often gets reflected by walls, and because of that, a lack of direct line of sight would not prevent a Cricket from reporting distance measurements[10]. Hence, in order to be able to detect direct line of sight, we had to add additional hardware to our system.

We decided to use IR LEDs and IR remote control sensors for this purpose. Every Mark III robot was equipped with 8 IR LEDs and 4 IR sensors. All new pieces of hardware were positioned in directions parallel to the ground, with the LEDs being 45^o apart and the sensors being 90^o apart. Figure 5-5 is a photograph of a Mark III robot with its new equipment.

In our mini-system, whenever two robots have to check whether there is a direct line of sight between them, one of them turns its LEDs on and the other reads its IR sensors. If at least one of the sensors reads a high enough value, then a line of sight is

[10]These measurements would be wrong, however, as they would equal the distance travelled by the ultrasound, which is different from the straight-line distance.

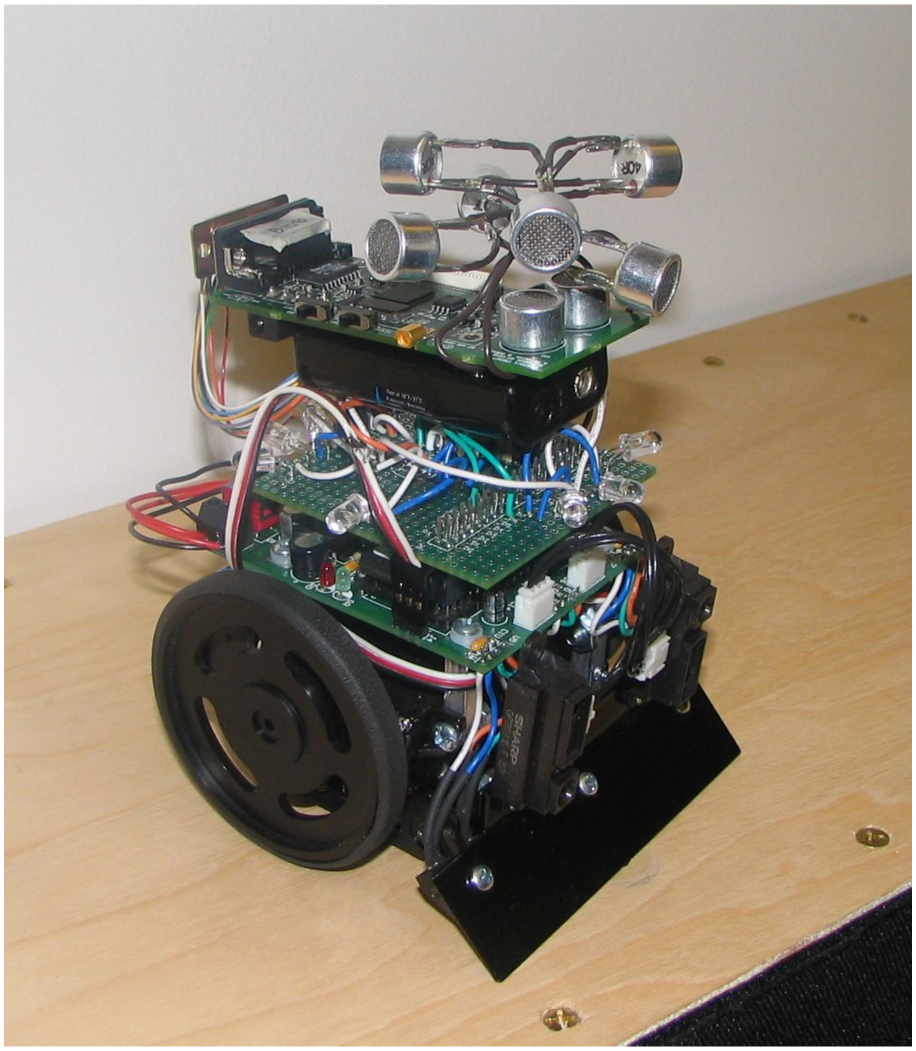
Figure 5-6: A coupled pair of a modified Cricket and a modified Mark III robot.

detected. Otherwise, the line of sight is pronounced interrupted. Allowing the robots to take a number of readings (and aggregate them) before they decide, reduced the errors of this subsystem to zero, at least to the extent to which we could observe it. This setup clearly satisfied the line-of-sight detection assumption of our model.

Finally, the wall-following ability did not need any additional hardware, as the Mark III robots are already equipped with a pair of frontal IR sensors, capable of measuring the distance to the closest object in front. This allowed us to create a simple differential controller, which reads the two sensor readings and performs wall following.

The detection of corners was implemented as follows. While a robot performs wall following, its Cricket would collect a series of coordinates (relative coordinates to two other Crickets, which are stationary at the time). The Cricket decides that a corner has been reached at the moment when it cannot fit all data points on one thick line (with the thickness being the margin of error of the distance-measurement system). This guarantees that no points between the wall and the line between the starting and final positions of the robot would be left out of the formed triangle.

Overall, based on the constructed proof-of-concept mini-system, we can conclude that the modified low-cost hardware units are capable of meeting the assumptions of our model in section 5.1. Thus, we conclude that the hardware requirements of our system are realistic for a low-cost distributed system.

5.7 Discussion of Reliability

Since we do not have a full-scale, real-world implementation of a system running the DFS Exploration Algorithm, it is hard to give an objective evaluation of the reliability of our proposed system. However, there are some weaknesses and theoretical problems that everyone attempting an implementation of this system should be aware of, and so it is important to discuss them and address them in advance, even before seeing any of them in practice.

On the positive side, our experience with the Crickets and the Mark III robots suggests that the communication, distance-measurement and navigation assumptions are realistic and can be met with relatively cheap hardware. The same holds for the wall-following and line-of-sight detection abilities as well. The latter two naturally need more effort to be made reliable, but it is nevertheless a realistic goal.

5.7.1 Failure Detection

As far as we can see now, there are two major issues with the assumptions outlined in section 5.1 above. The first one is the reliance on the robots being reliable (i.e., assuming they do not fail). In order for a distributed system to be cost-effective, it has to be built out of relatively cheap hardware. This means that every single robot has a higher probability of failing (as compared to a more expensive robot) and in addition, we have more robots, so the probability of any one of them failing is much higher. Robots are inevitably going to fail and we need to design our system for this.

In order to enable an implementation of our algorithm on a team of failure-prone robots, we propose the usage of failure detectors. That is, the system should have some clear mechanisms for checking the liveness of every robot, and for detecting and verifying failures. One way to provide this is to assume every robot is capable of checking the liveness of its own subsystems. Given that robots always move in pairs, triplets or larger groups, this should not be hard (as robots always have someone to check their distance-measurement and line-of-sight systems with). Once we have this assumption in place, we can have the decision-making robot periodically send "health queries" to all other robots. If a robot X responds with a message indicating the detection of a failure, or if X does not respond at all, then the decision-making robot will pronounce X failed and will assign another robot in X's place.

A more interesting problem is how to detect when the decision-making robot fails. For that purpose we can apply one of the many commonplace protocols used in distributed systems. Since we do not expect Byzantine failures, this should not be a serious challenge. For example, we can order the robots by their unique ID numbers and define the robot with the lowest ID number to be the decision-maker. Then every robot will have the responsibility of checking the liveness of its preceding robot (in that order). Thus, robot number 2 will check number 1 (the decision-maker), number 3 will check number 2, etc. Whenever a robot X detects the failure of its predecessor Y (that is, if X cannot communicate with Y, but can communicate with many other robots), X will announce the failure of Y and Y will be removed from the ordered list. In this case, X will overtake Y's predecessor, or it will become the decision-making robot in case there are no more robots with a smaller ID number (i.e., in case Y was the decision making robot).

It is also important to make sure that the state of the decision-making robot is regularly "backed up" to other robots, in case that robot fails. This should not be a serious complication, however.

5.7.2 Consistency of Sensor Readings

The second problematic assumption in section 5.1 is the consistency of readings across robots and across time. Assuming uniform hardware and stationary environment is generally realistic. However, this is not enough to provide such consistency as the one assumed (i.e., uniform hardware does not mean consistent readings). The elements of the environment are in the continuous world and discretizing them consistently, in a worst-case scenario, is theoretically impossible. For example, consider an obstacle boundary that has a sharp, but not ideally sharp, convex turn. Robots passing by this point need to present the relatively sharp curvature as a part of a polygon. Assume, for example, that there are two ways to represent this: one is to have a single vertex, the other is to have two vertices close to one another. Clearly, if the corner is ideally sharp, a single-vertex representation would be agreed upon by everyone. Also, clearly if the "corner" has a substantial width, a double-vertex representation would be preferred by every robot. However, where exactly is the threshold?

Say the threshold is at some corner width t, and the error of measurement is some small ϵ. Now, if the actual width happens to be closer than ϵ to t, then some robots will get values above the threshold and some will get values below the threshold. Thus, in this worst case, one plausible scenario would be to have one robot pass by this corner and build the topological map of C with one vertex there. Then when another robot comes by, trying to follow the map built by the first robot, the second robot sees two vertices instead. This would clearly disrupt the correct execution of the algorithm, especially in cases without distance measurements, such as walking over a series of type 3 segments, as the second robot would essentially get lost on the C map.

Our proposal for addressing this problem is the following:

1. We should try to get as accurate sensors as possible. If the hardware is precise enough, then such "too close to the threshold" situations may be extremely rare. For example, modern processors are known to have the vulnerability of falling into a meta-stable state when converting asynchronous signals into synchronous

ones, but the probability of this happening is so low that it is practically never observed.

2. Errors due to inconsistent readings can be corrected by a system, which revises previous readings, based on later observations. In our example from above, the second robot should be able to see that its reading is close to the threshold, so later on it will look at the shape and distances of other vertices and triangles, as well as the positions of other robots, and it will most likely be able to tell which was the "correct" (i.e., consistent with the other robot's) reading among the two near the threshold.[11]

Overall, the implementation of the system presented in this chapter should be difficult, but in any case not impossible. A small-scale real-world implementation, hopefully coming in the near future, would provide great insights into the applicability of this system design for solving Autonomous Area Search problems in the real world.

[11]A similar technique is thoroughly discussed in [5] and in many other papers addressing the SLAM problem.

Chapter 6

Future Work

The field of distributed robotic systems is in its infancy. However, as we discussed in chapters 1 and 2, the booming availability of standardized, high-quality hardware is likely to cause rapid growth of the research field in the next few years. Thus, in the near future, we expect to see numerous possibilities for pushing the scientific frontiers in the field, both in theoretical and in experimental work. Part of these possibilities, those opened up by this thesis, are presented in this chapter.

This thesis presents ground-breaking work in two areas:

- Theoretical, asymptotic analysis of performance in distributed robotic systems.
- Applying distributed robotic systems to problems, which are unsolvable in a non-distributed fashion.

This ground-breaking work opens up many possibilities for future research, which we classify in the following sections.

6.1 Location-Aware System

Theoretical Results

In section 3.5.2 we prove a bound of:

$$\frac{N}{R} + O\left(\frac{C}{R} + B\right)$$

for the time taken by our proposed Compact Coverage Algorithm. In section 3.3 we also express our opinion that this is also a fundamental lower bound for this problem. However, we only managed to prove a lower bound of:

$$\Omega\left(\frac{N+B}{R}+D\right)$$

Closing this gap between the proven bounds is a very interesting theoretical problem. If our hypothesis for the optimality of our bound is true, then there must be a fundamental difference between the capabilities (in terms of optimizing run time) of map-aware systems[1] and map-oblivious systems. If that is the case, then it is highly likely that research in the area of cache-oblivious systems [4, 18] would provide useful insights.

Addressing Harder Models

Our algorithm is designed under a set of assumptions that might not be valid in some scenarios. For example, we assume that the communication system is generally performing very well. For the purposes of theoretical analysis, we have encapsulated the total cost of bad communications in the C variable. However, as discussed in section 3.3, this value is not exogenous, and it actually depends on the specific algorithm. Because we assume that C is going to be small in any case, we do not try to optimize our algorithm with respect to it.

However, if we are to solve location-aware Distributed Area Search in a setting with an inferior communication system, the C factor might be quite substantial, and it might be very beneficial to design an algorithm which can tolerate inferior communication systems. One general guideline in this direction is to design an algorithm which attempts to keep robots at a distance from one another, whenever possible. This way, communication delays will not affect the number of covered cell repetitions, as robots will have a certain "margin" of time before they can possibly repeat one another's work.

[1]That is, knowing the map of the area in advance.

Another assumption relaxation worth considering is allowing Byzantine failures, instead of just stopping failures. One can imagine military applications in which the hijacking of a robot (or a few robots) by the enemy should not be able to disrupt the operation of the rest of the system. There are many well-known general purpose solutions to the Byzantine failures problem, but it would be interesting to see whether these can be applied to distributed robotics and especially to Distributed Area Search. Moreover, even if these solutions can be applied, evaluating their performance and efficiency implications would be an interesting direction for research. It is not at all inconceivable that some algorithms might perform better (in terms of run time) than others in the face of Byzantine failures, especially if we combine this with inferior communication system scenarios.

Allowing Broader Robot Base

The Compact Coverage Algorithm is designed for up to about 30 identical robots. While this might be sufficient in most scenarios, we might want to loosen this restriction, both in terms of diversity and scale.

It is easy to see the advantages of heterogeneous robotic teams. For example, if we have an environment with both substantial wide-open areas and narrow cluttered areas (similar to the Stata Center floor plan on figure 3-10), having robots of different size and mobility has obvious advantages. Large robots would perform much better in wide open areas, while smaller robots would be much better in finer-granularity areas.

However, once we have such a heterogeneous team, we must run a good algorithm that will utilize the advantages of the different robots to the fullest extent. Clearly, running CCA on a heterogeneous set of robots is far from the right thing to do. Thus, exploring this extension of the problem would also be an interesting piece of research.

Another case in which we might want to go above and beyond the limitations of the CCA, is if we are working on a very large scale with a very large number of robots (e.g. on the order of 100 or more). In such a setting the discrete CCA algorithm[2] should still

[2]This is the part of the system that was tested on the Java simulation in section 3.6.

perform well. However, the communication protocol described in section 3.4 would not be able to function under the growing pressure on the communication system. This is especially true in scenarios in which the messages in the communication system travel on multiple hops, as then the rate of total messages sent within the system per unit of time might grow quadratically on R.

In order to address such issues, we need a communication protocol that scales with R (i.e., whose steady-state time complexity is independent of the value of R), but which nevertheless keeps C low. This is definitely a hard problem, but it is certainly not unsolvable. A good approach to this problem would be to try to develop a protocol which communicates information immediately only to close neighbors, while sending distant neighbors only aggregated pieces of data[3], gathered by the robot and its near neighbors.

Building a Larger-Scale Real System

The experimental work on the location-aware system has a lot of potential for improvement. The Crickets are theoretically limited by a 10-meter range, but in practice we were unable to get our localization system to work often and accurately enough on an area larger than about 5 meters by 5 meters.

This size of the operational area might be greatly improved if, instead of only three stationary Crickets, we set up a whole grid of such “beacons”. If we use 25 of them, arranged in this fashion, and if they manage to compute their coordinates with small enough errors, then the robots should be able to compute their coordinates in an area as large as at least 400 square meters. It would be very interesting to see the results of such a larger-scale real-world experiment.

In fact, with a localization system of a larger scale, it would even be possible to build an application running the CCA. One should be able to connect Crickets to Roomba robotic vacuum cleaners (over their serial ports) and build a distributed cleaning application. Also, if one can find or build appropriate robots for this, a distributed lawn-mowing application, built out of GPS-equipped units, would likely

[3]This might be encoded in a Quadtree, or maybe in some other similar format.

be possible as well.

6.2 Autonomous System

Efficiency and Fault-Tolerance

This thesis makes a first attempt at addressing the Distributed Area Search problem under an autonomous assumption, with the requirement for 100% guaranteed correctness. While the results we achieve are very important as a basis to start thinking about the problem in this fashion, they are far from enabling the practical implementation of a high-performance real-world system.

In order to address these practical issues, one needs to set fault-tolerance and efficiency as goals of the system and to then address those goals appropriately. In chapter 5, we base our system on a single thread of control and complete serialism, but this does not have to be the case for every autonomous solution of Distributed Area Search.

The DFS Exploration Algorithm has many properties, which suggest that it would be relatively easy to make it work in a parallel fashion, where not only the coverage, but also the control is distributed. The various "front line" expansions are essentially independent from one another. Moreover, every front line between covered and uncovered areas is marked by the presence of a pair of robots, hence every robot can clearly distinguish between the case when it is expanding these front lines and when it is not. Thus, a parallel system running our algorithm would not suffer from data synchronization problems as many distributed, parallel systems do.

Furthermore, the above is also likely to provide a better basis for assuring fault-tolerance in the system, another very important goal from practical perspective.

Also, doing more real-world experiments with actual hardware would surely provide better insights into the issues of building a real autonomous Distributed Area Search system, as well as allow us to test the efficiency and/or fault-tolerance properties of any parallelized version of the DFS Exploration Algorithm.

Theoretical Results

Finally, there are many interesting theoretical problems under the autonomous system setting. The first of them is formally proving the complexity bounds of our algorithm, as outlined in section 5.5. Another interesting and very challenging project would be proving that these bounds are fundamental lower bounds to the problem and that no other algorithm can do better. Alternatively, improving these bounds, or providing an algorithm that solves the problem with fewer robots would definitely be a great extension of our work.

The theoretical problems become even more interesting once we set efficiency and fault-tolerance as our goals. Giving a bound on the time taken by a distributed-control system, implementing a parallel version of the DFS Exploration Algorithm, would be an exciting and certainly non-trivial problem. Proving a fundamental lower bound for this case would be even harder, but also very rewarding from theoretical perspective.

Chapter 7

Conclusion

In this thesis we demonstrated the applicability of the distributed systems paradigm to the field of robotics by presenting solutions to the Distributed Area Search problem under two different models.

Our first solution, the Compact Coverage Algorithm presented in chapter 3, works under the assumption of the presence of a global localization system and provides two important properties of distributed systems: efficiency and fault-tolerance. The efficiency of the CCA is proved to be near-optimal by a tight asymptotic upper bound on its run time. The excellent practical performance of the algorithm is demonstrated through a series of comparative test runs on a Java simulation. The realism of the assumptions of the CCA is demonstrated through a prototype system, implementing the algorithm on a team of twelve robots.

To the best of our knowledge, our work on the CCA is one of the first serious attempts to combine the fields of distributed systems and robotics by applying distributed systems style analysis to a robotic system. Unlike virtually all research in robotics, we present theoretical bounds on the scalability and asymptotic performance of our system. This work, in our opinion, can serve as a starting point for developing a more quantitative branch in collaborative robotics, in which distributed robotic systems are analyzed with the same rigor as distributed computer systems.

Our second solution to Distributed Area Search, the DFS Exploration Algorithm presented in chapter 5, works under a completely autonomous model, requiring no external infrastructure. We demonstrate that, in a worst-case scenario, the Area Search problem is unsolvable under this model, unless we take a distributed approach. We also present a distributed algorithm solving the problem, thus proving that it is in fact solvable with a large enough[1] team of robots.

The DFS Exploration Algorithm is only a ground-breaking first attempt at tackling Distributed Area Search under the asynchronous model. It is only concerned with the correctness of the solution, but it also establishes the basis for and opens the possibilities of creating efficient and fault-tolerant solutions to the problem.

Based on the two solutions to Distributed Area Search presented in this thesis, we can conclude that distributed systems have an enormous applicability in the field of robotics, both in terms of allowing more efficient and reliable solutions, and in terms of allowing us to solve otherwise unsolvable problems in the real, physical world.

[1]But nevertheless reasonable in size.

Bibliography

[1] W. Burgard, D. Fox, and S. Thrun. Active mobile robot localization. *IJCAI*, 1997.

[2] W. Burgard, M. Moors, D. Fox, R. Simmons, and S. Thrun. Collaborative multi-robot exploration. *IEEE ICRA*, 2000.

[3] The Cricket indoor location system. http://cricket.csail.mit.edu/.

[4] E. Demaine. Review of the cache-oblivious model. http://theory.csail.mit.edu/classes/6.897/spring03/scribe%5Fnotes/. Notes for MIT Course 6.897: Advanced Data Structures.

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

[6] D. Gay, P. Levis, R. von Behren, M. Welsh, et al. The nesC language: A holistic approach to networked embedded systems. *ACM SIGPLAN*, pages 1–11, 2003.

[7] B. Gerkey and M. Mataric. Sold!: Auction methods for multi-robot coordination. *IEEE Transactions on Robotics and Automation, special issue on Advances in Multi-Robot Systems*, 18(5):758–786, October 2002.

[8] C. Grinstead and J. Snell. *Introduction to Probability*, chapter 12. American Mathematical Society, 1997.

[9] D. Guzzoni, A. Cheyer, L. Julia, and K. Konolige. Many robots make short work. *AI Magazine*, 1997. SRI International / 1996 AAAI Robot Contest.

[10] R. Kurazume, S. Nagata, and S. Hirose. Cooperative positioning with multiple robots. *ICRA*, 1994.

[11] K. Lerman, C. Jones, A. Galstyan, and M. Mataric. Analysis of dynamic task allocation in multi-robot systems. *International Journal of Robotics Research*, 25(3):225–242, March 2006.

[12] C. Luo and S. Yang. A real-time cooperative sweeping strategy for multiple cleaning robots. *IEEE Int. Symposium on Intelligent Control*, pages 660–665, 2002.

[13] Mark III OOPic version. http://www.junun.org/MarkIII/Info.jsp?item=27.

[14] E. Martinson and R. Arkin. Learning to role-switch in multi-robot systems. *IEEE ICRA*, 2003.

[15] M. Mataric, M. Nilsson, and K. Simsarian. Cooperative multi-robot box-pushing. *IROS*, 1995.

[16] J. McLurkin. Stupid robot tricks. Master's thesis, Massachusetts Institute of Technology, 2004.

[17] T. Min and H. Yin. A decentralized approach for cooperative sweeping by multiple mobile robots. *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 1:380–385, 1998.

[18] Harald Prokop. Cache-oblivious algorithms. Master's thesis, Massachusetts Institute of Technology, 1999.

[19] I. Rekleitis, G. Dudek, and E. Milios. On multiagent exploration. *Vision Interface*, pages 455–461, June 1998. Vancouver, Canada.

[20] I. Rekleitis, V. Lee-Shue, A. New, and H. Choset. Limited communication, multi-robot team based coverage. *IEEE ICRA*, pages 3462–3468, 2004.

[21] The Roomba robotic vacuum cleaner. http://www.roombavac.com/.

[22] R. Simmons, D. Apfelbaum, W. Burgard, D. Fox, M. Moors, S. Thrun, and H. Younes. Coordination for multi-robot exploration and mapping. *AAAI/IAAI*, 2000.

[23] S. Thrun, S. Thayer, W. Whittaker, C. Baker, W. Burgard, et al. Autonomous exploration and mapping of abandoned mines. *IEEE Robotics and Automation Magazine*, 2004.

[24] TinyOS. http://www.tinyos.net/.

[25] P. Ulam and T. Balch. Niche selection in foraging tasks in multi-robot teams using reinforcement learning. *2nd International Workshop on the Mathematics and Algorithms of Social Insects*, 2003.

[26] G. Welch and G. Bishop. An introduction to the Kalman filter. http://www.cs.unc.edu/%7Ewelch/kalman/kalmanIntro.html.

[27] B. Yamauchi. Frontier-based exploration using multiple robots. *ICAA*, 1998.

[28] B. Yamauchi, A. Schultz, and W. Adams. Mobile robot exploration and map-building with continuous localization. *IEEE ICRA*, 1998.