Swarm Dispersion via Leader Election, Potential Fields, and Counting Hops

A THESIS
SUBMITTED TO THE FACULTY OF THE GRADUATE SCHOOL
OF THE UNIVERSITY OF MINNESOTA
BY

Anuraag Pakanati

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF SCIENCE

Maria Gini, Advisor

March 2010

Acknowledgments

I would like to express my sincere appreciation towards my advisor, Dr. Maria Gini, who not only embraced me as a member of her research team, but has been exceedingly patient and always prepared with sage advice throughout this research. I also owe thanks to my fellow students at Minnesota, especially Steven Damer who provided some very helpful assistance in the area of robot communication. Lastly, I am grateful to those students whose work here at Minnesota preceded mine, including Ryan Morlok and particularly Luke Ludwig, who, although I never met them, left behind useful techniques, tools, and code that proved extremely helpful in beginning my research here.

I would also like to thank my parents, Shraven Pakanati and Sree Pakanati, my sister, Sheena Pakanati, and Elizabeth Duval for their forbearance, assistance, and editing throughout this process.

Abstract

Robot swarm dispersion is an important area of research with many applications including the deployment of sensor networks for military, search and rescue, emergency response, intruder alert, and other applications. In this work, we examine how robots with only laser data and the ability to communicate within a certain radius may disperse in a complex unknown environment. We present a python framework for running and evaluating robot swarms using controllers within the Player/Stage framework. Finally, we demonstrate a modular state-based algorithm for robot dispersion which attempts to maximize coverage while minimizing loss of connectivity during deployment. We demonstrate using the Player/Stage environment that this can in fact reliably achieve over a 100% percent increase in coverage over a variety of different environments.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Consider a domain such as a cave, an unexplored building, or disaster area in which the layout may be unknown ahead of time. It may be desirable to construct a wireless network from sensor nodes in this domain in order to facilitate the deployment of men and material, or perhaps gather and transmit sensor information. Automated methods to accomplish this are highly desirable since not only do they avoid placing humans in harm's way, but they allow human resources to be focused, either elsewhere or more precisely with information gathered by such a network.

Even when a map is available beforehand, finding an optimal deployment is a nontrivial problem. If one were for example to divide a 100' x 100' space into (very large) 1' x 1' grid segments, there would be 10000 possible locations. Placing just four robots on that would require searching through $\binom{10000}{4} = 99{,}958{,}003{,}799{,}940{,}000$ possible robot placements with a brute force search approach. Although there are many optimizations to drastically reduce the search space, finer placements, more robots, and larger spaces all make the problem much more difficult. To the best of our knowledge, no one has managed to find an algorithm to find an optimal placement on an arbitrary continuous domain. Because of this, solving the problem of finding a deployment that maximizes coverage in a complex, unknown environment, must necessarily reduce to finding an approximate solution. Within the context of discretized approximations to the continuous problems in known spaces, some groups have managed to find optimal algorithms for coverage as in [20].

We approach this problem by building upon the technique of potential fields, first introduced by Khatib in [27]. Specifically, we seek to use connectivity to modify the potential fields in order to achieve dispersion with an eye to preserving certain network properties. Fundamentally, we take an empirical approach due to the extreme difficulty of providing theoretical guarantees on the performance of such systems. There are similarities between the potential fields and the n-body problem in mathematics, so it is conceivable that chaotic robot paths would in fact be the rule, rather than the exception, thereby complicating theoretical analysis.

1.1 Problem Description

Robot swarm dispersion refers to a broad class of problems in which a collection of robots start out in a relatively compact space and must spread out in order to cover as much area as possible while preserving connectivity of the network. Applications of the dispersion problem include search and rescue and emergency response, particularly in instances in which an environment is not fully known, time is critical, or the environment is hazardous, security, tracking, area monitoring, and wireless network construction.

There are two broad approaches to the problem of dispersion: one in which the environment is mapped as you explore, and model-free approaches. In this work we focus on model-free approaches and do not attempt to characterize the environment. This is because we are interested in studying dispersion with smaller, less capable, and thus more inexpensive robots. In our work, robots lack GPS, maps, or any form of absolute localization, but do have access to laser range-finders and wireless communication. They must use this limited information from their sensors to globally achieve a "good" arrangement in a distributed fashion. Thus, approaches such as Simultaneous Localization and Mapping (SLAM) are unfortunately outside the scope of this work. We refer the interested reader to an excellent survey paper on mapping by Sebastian Thrun in [41].

It is first important to note that no universal definition of an 'optimal configuration' exists, since different applications may emphasize different characteristics of a swarm deployment. For example, robustness to node failure may be an important attribute in a tracking application such as radar, but less important in some other applications. Some metrics for the effectiveness of dispersion algorithms involve evaluating the amount of coverage achieved by the network at a certain time, the rate of coverage increase over time, the time that it takes for the network to be deployed, robustness to node failure, and the degree of 'overcoverage', among others. As mentioned earlier, finding an optimal set of locations to maximize robot coverage within a continuous domain is itself a combinatorially expensive problem. Thus, the objective must be to find an approximation of optimality, particularly in a system in which this is being measured in real time.

1.2 Overview

In this work, we present three major contributions:

1. we present a novel algorithm to enable dispersion of a group of minimally equipped robots in an unknown environment. The algorithm uses leader-election, hop counting, and potential fields to maximize space coverage while maintaining the robots within communication range of each other. The robots do not need precise odometry. They rely only on a laser range finder they use to sense the surrounding environment and a wireless card they use to stay connected with the other robots.
2. We describe a framework implemented in python for running and evaluating robot swarm dispersion using the Player/Stage robot simulation environment. The framework enables easy reconfiguration of the experiments via XML files, provides multiple analysis tools for data collection and analysis, and includes a variety of robot behaviors that can be used to build dispersion algorithms.
3. We provide extensive experimental results obtained in simulation in a variety of environments, with different parameters. The experimental results show the effectiveness of the algorithm which achieves a significant increase in coverage while maintaining connectivity.

In Chapter 2, we discuss some of the background literature related to our work. In Chapter 3, we discuss the simulation environment and our methods and why they sometimes did not perform well. In Chapter 4, we discuss the results of some preliminary experiments that we tried using nearest-neighbor approaches. Chapter 5 contains a description of three sets of experiments, where we measure our algorithm on four environments, measure the effect of sensitivity, and test the effects of sensor degradation on algorithm performance. Finally, in Chapter 6, we wrap up with a conclusion and some discussion of future work.

Chapter 2

Related Work

Although there is a relatively small amount of work specifically devoted to the particular problem of robot dispersion as defined above, the problem is nonetheless closely related to many other interconnected topics in robotics research, including coverage, spanning trees, pheromone-based approaches, incremental approaches to coverage, centralized approaches, potential fields, wireless localization, sensor networks, and robot behavior.

2.1 Definitions

In [15], Gage outlined a taxonomy of coverage problems: blanket coverage, which involved maximizing the coverage area of a static configuration, barrier coverage, which seeks to maximize the probability of detection through a barrier of robots. Finally, there is sweep coverage, in which the general idea is to maximize the overall number of detections in a region, usually when static coverage approaches are not practical. What Gage refers to as 'sweep coverage' is also sometimes called the dynamic coverage problem as in [7]. Within the context of Gage's framework, the dispersion problem can be described as a blanket coverage problem with a connectivity constraint.

2.2 Sensor networks

Sensor networks refer to a broad area of active research devoted to the study of wireless sensor networks deployed for a variety of reasons, including area monitoring. These are often, but not necessarily, immobile nodes distributed over a region and which combined constitute a communications network. Much, but not all, of the research in the sensor network community is rooted in the assumption that node placement is final and that the problem then becomes optimizing available resources to reduce power consumption while maximizing coverage. The sensor network literature often tackles such topics as localization, particularly in the context of airdropped or deployed nodes that do not necessarily have any apriori information about their location or relation to the remainder of the internet. While much of the literature is intimately linked with the dispersion problem, we are particularly interested in the subsets of sensor network research that utilize mobile nodes.

In [13], Silva et al. use blind sensor networks combined with a centralized topological analysis to measure coverage. Specifically, given certain environmental properties and no localization or orientation information, it is possible to determine coverage in an unknown domain. While this is outside the scope of this paper, it is nonetheless a fascinating example of complex calculations that can be done with very little information available to a robot.

2.2.1 Localization

The interesting result that variation in indoor WiFi signal is limited to about 10 dB was demonstrated by Howard et al. in [24]. Additionally, the same group found that signal strength is comparable across robots with identical hardware and that orientation has a limited effect on signal strength. This result may not generalize well to smaller signal sources, but nonetheless has implications for our assumption that signal strength degrades uniformly with respect to the inverse of the distance squared.

An additional approach was taken by Haeberlen et al. in [18]. They collected 802.11 wireless signal intensity measurements for cells corresponding to offices and hallways to nearby base stations. Larger rooms were broken up into smaller areas for the purpose of these measurements. Signal intensities were then fit to a normal distribution. They used a Markovian approach to use signal strength and estimate present location to the cell unit. Although this approach does not permit localization within a couple of meters, it does have the distinct advantage of utilizing existing base stations in order to localize.

Another approach to wireless localization in a randomly placed sensor network is undertaken by Bachrach et al. in [5]. They assume that there exists some subset of nodes called seeds that have access to GPS or some other position information.

These seeds propagate messages to all neighbors which, then in turn propagate these messages outwards. A counter is maintained to prevent backpropagation of the message. Nodes estimate their position through an incremental update using gradient descent. The result of their approach was a local coordinate system up to within 20% of the local radio range even with slight variation in the actual radio ranges. However, they point out that the algorithm may overestimate the distance when sensors are arranged around a large hole.

A different localization algorithm is used by Akcan et al. in [3]. They utilize nodes equipped with a compass and range measurement device are able to use this information to follow a directed trajectory. This obviates the need for an external or absolute localization source, such as that provided by GPS, although the compass still provides an absolute orientation measurement, albeit with error. Additionally, in indoor environments, a compass may be susceptible to giving incorrect readings because of metal inside walls. Our own approach does not attempt localization in any way, but it may be useful in the context of directed dispersion extensions over our approach.

Langendoen et al. in [28] characterize distributed localization as generically having three phases: (1) determine the node-anchor distances, (2) compute node positions, and (3) refine the positions through an iterative procedure. They evaluate several different combinations of known algorithms and found that no single combination performed best in their simulation environments and that the best choice of algorithm for a given case was determined by range errors, connectivity, anchor fraction, placement, and other considerations.

2.2.2 Mobile Wireless Sensor Networks

Yuan et al. in [43] conducted a review of control and localization literature in mobile sensor networks. They define topological control as having the goal of setting radio ranges for each node to minimize energy usage and coverage control as the problem of deploying nodes to maximize coverage. Our problem thus more properly falls under the category of coverage control, specifically static control, since we seek to deploy a network to cover as much of the environment as possible, while maintaining connectivity. Nonetheless, many of the state-of-the-art methods from topological control are relevant to our work– our algorithm is a semi-centralized approach using both leader election and heavily influenced by k-connectivity ideas.

In [7], Batalin et al. introduce an approach they call the Molecular Technique. This approach does not use any communication and relies only on vision. Robots may detect beacons atop other robots and utilize this information to disperse in different directions in order to perform a coverage task. Our own approach bears some similarities, however the visual input that we have does not provide any robot position information– merely distance to objects, and we have a connectivity constraint that limits robotic movement.

2.3 Centralized and Semi-centralized Approaches

Not all approaches to the dispersion problem have been decentralized in nature. Below are a small sample of approaches that do not use a purely decentralized implementation. Our own work also falls in this category– it relies on an emergent centralized behavior to limit connectivity losses.

Parker et al. introduce a novel approach for mobile sensor network deployment with heterogeneous robots in [35]. In this study, two helper robots, a leader, and a follower, with several capabilities "herd" mobile sensor nodes with no ability for localization or obstacle avoidance into deployment positions. They do this using line of sight and the follower teleoperates the mobile sensor nodes to guide them to their destination.

Another approach to robot control, with the goal of mapping an unexplored building, detecting and tracking intruders, and transmitting that information to a remote operator is presented by Howard et al. in [23]. In order to accomplish this, the authors construct a heterogeneous team of roughly 80 robots, with a small number of highly capable and expensive robots, and a large number of relatively simple robots. In the first phase, simultaneous localization and mapping is used to map the environment. A centralized location is used to integrate the data from all the robots and construct a map and pick desired sensor locations. In the second phase, capable robots, followed out by the sensor robots travel to each desired sensor location and deploy the sensors in turn. The sensor robots then constitute a sensor network at the end of this process and the larger robots may be reused.

Pearce et al. in [37] utilize a virtual pheromone approach to study the problem of dispersing Scout robots from an external centralized human or robot controller. Scouts are small mobile platforms measuring roughly 11 cm long by 4cm wide. Their small size and limited computational power lead the authors to implement a centralized approach, whereby another robot or controller calculates a desired path and communicates that information to each Scout. Scout location and orientation were calculated using a blobfinder device on the controller. This algorithm has the advantage of forcing the scouts into the locations one would expect from a regular polygon.

Centralized approaches offer some benefits over distributed approaches. One is that the capabilities of the smaller robots need not be as strong as those of the larger robots, leading to large potential cost-savings in deployment. Indeed, as Howard et al. in [23] point out in their analysis, heterogeneous systems by their nature may force algorithmic controller decisions to be platform-dependent. Additionally, another consideration that must be made when contrasting the two approaches is that centralized algorithms introduce centralized points of failure. If a controller robot were to fail, the entire dispersion attempt might fail as well. This can be mitigated to some extent, depending on the capabilities of the small robots, but it must remain a consideration, when deploying swarms of robots with limited capabilities.

2.4 Incremental Deployment

As with all approaches, there are advantages and disadvantages to incremental approaches. One advantage would be that far fewer resources would be required to achieve the same result in terms of cost. Such approaches have the obvious drawback of requiring far more time, since a single robot must place all the nodes by itself. Additionally, it would be very difficult for such a network to reconfigure itself, which is one of the main advantages of using robots as sensors. There are two broad classes of incremental approaches to the coverage problem. The first could be classified as deployment-based approaches whereby one robot deploys immobile sensor nodes. This contrasts with the approach of using mobile motes to deploy sequentially.

2.4.1 Sequential Deployment of Immobile Nodes

The LRV algorithm proposed by [9] has a single robot robot carrying around network nodes in an unexplored area and deploying them in a rough lattice, depending on local conditions. It utilizes these sensors for navigation instructions to the least recently visited area. In some respects, this bears some resemblance to both the pheromone approaches and the potential field approaches described later in Section 2.6.1. They present a similar approach in [8] using value iteration performed by the deployed sensor network to assist the robot in navigating between points.

2.4.2 Sequential Deployment of Mobile Nodes

Another sequential approach is taken by Howard et al. in [21] where nodes are added incrementally with one of four different algorithms with a requirement that nodes be within visible distance of each other. The first algorithm adds a node randomly in reachable space, where reachable space is defined as within visible range of a node already in the network. The second algorithm adds a new node on the boundary point between reachable and unreachable space, the third algorithm adds a new node where it will maximize coverage as defined by visible space, and the fourth algorithm finds the boundary point that will maximize coverage. Nodes can be swapped in function if a node attempting to deploy is obstructed. They compare their results against a greedy incremental approach in which the nodes are placed at the location that maximizes coverage without regard to the mechanics of moving there. We will be using the same measure in order to have a baseline for our own dispersion results for comparison.

2.5 Distributed Approaches to Coverage

2.5.1 Spanning Trees

Spanning trees are a general approach to solving coverage problems in known and unknown environments. The world is first divided into cells of equal size and shape which [12] categorizes as approximate cellular decomposition. [14] introduced the Spanning Tree Coverage (STC) algorithm in three forms: an offline implementation of STC, an online versions of STC that use sensor data, and an ant-like STC that uses pheromones. The offline STC algorithm takes a map of the environment and outputs a STC that a robot then uses to cover the domain by following the path in a counterclockwise manner. The online algorithm requires a position, orientation, and obstacle detection sensor to construct a STC. Finally, the ant-like STC again uses position, orientation, obstacle detection sensors, along with a marker detector, but only requires O(1) memory as compared to O(n) memory for the online algorithm, where n is the number of cells in the world.

Hazon et al. extended the idea of STC to a multi-robot setting in [19]. The problem may be summarized as given a STC, how can multiple robots be allocated in order to minimize the overall coverage time in a robust manner? The MSTC algorithms they provided were both for cases in which backtracking, defined as retracing over an already-covered area, was allowed and when it was not allowed. They showed that all MSTC algorithms carry the potential for a best case, in which robots evenly distributed the area to be covered. They also showed that the worst-case in a non-backtracking algorithm, which occurred when the robots were a space apart and thus did not divide the work evenly, had roughly the same covering time as a single robot. The backtracking MSTC they provided helped deal with this eventuality and ensured that the covering time would be at worst half that of a single robot.

In [20], the MSTC idea is extended to an on-line setting where the environment is not necessarily known beforehand and the robots must construct their paths incrementally as they discover new obstacles, terrain, and even each other. This approach assumes that the robots have a common coordinate system available to them. Each robot constructs a local spanning tree incrementally. The result is a robust multi-robot algorithm that spans the entire domain.

The problem of finding paths that minimize the total coverage problem for a given multi-robot configuration was investigated by Agmon et al. in [1, 2]. This is hypothesized to be a NP-hard problem. They assume that the robots have complete knowledge of the terrain, perfect odometry, and localization capabilities and access to the results. The results of this work show that the structure of the spanning tree greatly affects the performance of algorithms such as MSTC.

Spanning trees provide some theoretical properties that make them attractive, such as guarantees about coverage and coverage time. However, as a class of solutions they tend to make several assumptions that may not hold for many applications. An offline map is unlikely to be unavailable in most dispersion scenarios. Similarly, exact position data and odometry require expensive equipment, and are probably cost prohibitive in an application such as a robot swarm.

2.6 Distributed Approaches to Dispersion

There are several algorithms that researchers have developed in attacking the dispersion problem as above. This is by no means intended to be an exhaustive list of all the approaches that have been tried, but instead a somewhat representative sample.

The problem of dispersing a swarm of robots in an unknown environment was also tackled by Morlok et al. in [33]. To this end, they compared four different approaches: a random walk, in which robots moved forward with a small random turn factor that was updated at short intervals, a wall-following behavior which seeks to use obstacles to explore an environment, an algorithm seeking to move towards open areas by avoiding obstacles, and finally a fiducial behavior in which a fiducial device allowed robots to localize relative to each other and use that information to move away from each other. The fiducial behavior performed the best out of those sampled on a variety of domains, followed surprisingly by the random walk in their experiments.

The clique algorithm was introduced by Ludwig et al. in [30] and used wireless intensity signals to approach the problem of dispersion. It uses wireless signals to disperse robots, even though they do not know the relative locations of other robots. In order for this to work, some robots are designated as 'sentries' which do not move, yet allow other robots to vary their distance. Robots share connectivity information with each other so that they can all agree on a set of sentries by following a set of specific rules. They implemented four behaviors– guard in which robots stayed still, collision avoidance to steer away from walls and other obstacles, connection seeking to move backwards towards a sentry node in the event of a disconnection, and dispersion in which robots attempt to decrease wireless signal intensity to a sentry node. This approach had the distinct advantage of being very minimalistic– the components involved were cheap and well-suited to a swarm like approach.

Another approach to the dispersion problem using a chain or chains of robots entering from a single entry point was proposed by Hsiang et al. in [25]. They assume that the world is and that robots are finite state automata with O(1) memory. A robot may occupy one pixel at a time, and a pixel has three states visible to a nearby robot: occupied, unoccupied, or previously occupied. They proposed three algorithms– a depth-first leader follower algorithm for a single chain, in which the leader of the chain seeks frontier pixels that have never been occupied, a breadth-first leader follower algorithm again for a single chain in which there may be multiple leaders, and finally a laminar flow leader-follower algorithm for multiple chains. The laminar flow algorithm may take as much as log(k) * the makespan of an optimal algorithm, where k is the number of robots that can enter at every time step. Despite having desirable properties such as a theoretical bound on the coverage time, the general approach suffers from a few drawbacks. First, robots must agree on a shared coordinate grid, which is itself a nontrivial problem. Secondly, the ability to detect if a pixel had previously been covered, requires some kind of environmental marking ability which probably would involve pheromones and a device to detect pheromones. Additionally, the work assumes that there are enough robots available to cover the environment, which may not always be true. In our approach, we examine the problem where there are fewer robots available than the minimum amount to cover.

The uniform dispersion problem in an empty room was investigated by Ishigaki et al. in [26] by utilizing two landmarks set on the wall. They first perform alignment between the landmarks and then disperse using communication. This approach is not readily generalizable to many environments, since it is unlikely that common, identifiable landmarks will be visible or even available in any occluded environment.

A directed dispersion algorithm using swarm robots which were small rechargeable robots that cycle back to a charger was presented by McLurkin et al. in [32]. In this algorithm, robots designate themselves as interior, wall robots, or frontier robots. Interior robots are those that can detect neighbors in all directions. Robots that can detect a boundary will create a "virtual" neighbor and designate themselves as wall robots unless there is a segment of more than 180° without any neighbors, in which case they will designate themselves as a frontier robot. Similarly, frontier robots are all of those with more than 180° without neighbors. Using virtual pheromones similar to those proposed by Payton, a gradient is formed towards the frontiers. This gradient is followed by interior robots to travel towards the edges. In the system they present, this is an especially useful feature since robots must constantly return to the charger to recharge.

2.6.1 Pheromone Approaches

Pheromone algorithms describe a broad class of biologically inspired approaches to both dispersion problems. Ants and termites in particular are relatively simple organisms which are able to achieve breathtaking coordination in path-planning and dispersion by deploying chemical pheromones that help guide behavior. Several groups of researchers have attempted to apply the concept of pheromones in their work in the hope of duplicating the success achieved by insects. There are, however, inherent challenges in the use of chemical pheromones in their original use– namely that robots would have a limited supply of a substance, the chemicals may diffuse irregularly, and robots cannot detect many substances easily without relatively expensive equipment that would be prohibitive in a swarm approach. Odor detection is itself a very difficult problem, and olfaction can only currently detect certain types of odors [11, 29]. In addition to the inherent problems of detecting odors that pheromones involve, four unique properties of natural pheromones that make it difficult to mimic with artificial means were outlined by Sugawara et al. in [40]. Marking– pheromones must persist long enough to be detected by other agents, diffusion– a pheromone must spread over a large area, evaporation– pheromones should be volatile and not persist so that agents do not waste time and energy exploring old trails, and diversity– more than one pheromone might be required.

To avoid this problem, Payton et al. in [36] introduced the concept of virtual pheromones as applied to the problem of dispersion. In this article, robots are equipped with eight infrared digital receivers and eight infrared transmitting beacons. An originating robot initiates a pheromone field by specifying a 'hop-count' and sending messages that its neighbors then propagate in turn while decreasing the 'hop-count'. Their particular approach also draws on some of the same ideas described above in potential fields. The robots initially start clumped together and attract and repel each other based on the distance between them. Another interesting concept in their work is the idea of 'budding' whereby one robot is temporarily given a repulsive force that is stronger than its attractive force, which results in the robot attempting to get further away, while the other robots expand to fill the space.

Another pheromone-based algorithm for coverage was presented by Osherovich et al. in [34]. This was the Mark-Ant-Walk algorithm in which a robot marks a radius around themselves and then moves to the point with the least marking within (a slightly larger) visual range. It requires accurate sensing and odometry and glosses over the fact that marking is itself difficult. With that said, it provides some exceedingly good theoretical qualities; namely it is complete, offers bounds for efficiency, and provably distributes coverage evenly within certain bounds, all extremely nice properties for such a simple algorithm.

2.6.2 Potential Fields and Attract/Repel Approaches

In [27], Khatib introduced the idea of Artificial Potential Fields. The basic idea is that a robot exists in a field of forces. Obstacles act as repulsors, and in his original formulation, a goal represented an attractor. The robot calculates the sum of the force vectors acting on it to compute its trajectory. There have been various interpretations of this same basic idea, but at its core it has remained among the most influential ideas in mobile robotics. Potential fields have the advantage of being eminently computable at a local level– making them and derivations of them a highly popular choice for many distributed algorithms, including ours. They have the distinct disadvantage, however, of being highly prone to getting stuck in local minima. Additionally, potential fields can be highly jittery and chaotic in nature, with very small changes producing large differences in deployment and performance.

The possibility of using potential fields with bivariate normal fields to guide formation behavior was illustrated by Barnes et al. in [6]. These fields are derived independently from attractive points, obstacles which function as repulsors, and some behavior relative to the center of the swarm which induces a formation. They showed that varying the physical parameters of the robots in the swarm (but not necessarily the underlying behavior) still permitted this approach to work.

Two different attract/repel approaches based on local interaction were specified by Batalin et al. in [7]. The first one, in which robots are able to uniquely identify others, was called Informative. The second, in which robots are anonymous, was called Molecular. The Molecular approach that they proposed uses vision to detect other robots. They found that the Molecular approach was able to generate a solution within 5-7% of a manually generated optimal solution, where optimality was defined in terms of total coverage.

Another take on artificial potential fields was explored by Poduri et al. in [39]. Robots were treated as charged particles with two forces. One is a repulsive force that tries to maximize coverage. The other is an attractive force that attempts to maintain a degree K. They achieved good coverage and connectivity with high probability. The approach we take in this work is similar in that we also use the idea of connectivity to maintain an attractive force, although our work differs in that we combine this with hop-counting.

Howard et al. proposed a solution whereby robots equipped with 360° laser range finders used potential fields and a viscous friction term in order to deploy robots in [22]. This is perhaps the closest work to our own, in the sense that we also use 360° lasers with potential fields and are also working on a similar problem. The differences lie in our use of communication and connectivity to stop robot movement and prevent disconnection. The viscous friction term also bears some note, as potential fields as a class tend to be very jittery and sensitive to small changes which will frequently ripple throughout the system, which can complicate convergence. They introduce the viscous friction term in order to assure that the network converges. As they note, the fundamental assumption is that the environment is static– such a construct would not fare well in a dynamic environment. The results of their experiments showed that coverage increased over the run of the experiment, with the fastest expansion occurring initially and slower expansion still occurring when the experiments were terminated after three hundred seconds. Another interesting property of the final deployment was that the node placement was fairly even, despite the complete absence of explicit communication.

In [42], Ugur et al. investigated the use of potential fields by using indoor wireless signal modeling. They performed 16000 wireless measurements by varying distance and the orientation of both an originator and a recipient of a signal. The results of this empirical observation was that there was no easy way to fit the observed data using the physical characteristics of a setup, even when there were no boundaries involved. This highlights some of the difficulties in modeling the wireless signal modeling– for a boundary-less environment is without any doubt the most model-friendly. Notwithstanding that, it is not practical in many applications to go out and measure the wireless signal intensities beforehand, accounting not only for open space as they did, but also for the presence of walls or other obstacles. For example, the space may not be accessible, or attempting to get estimates beforehand may compromise the success of the operation.

Neighbor-Every-Theta graphs were introduced by Poduri et al. in which each node not on the boundary has at least one neighbor in every θ arc of communication range [38]. In order to achieve this, each robot requires relative position information of neighbors. These algorithms can be leveraged to obtain connected graphs where the interior nodes have a degree determined by the specified θ-value.

2.7 Our contribution

Our own work draws on several of these research threads. As mentioned earlier, we are interested in model-free approaches where robots do not attempt to map the environment in any way. Our robots have wireless connectivity and laser range-finding available to them and they must disperse from their initial positions to cover as much of the environment as possible without breaking connectivity. In relation to previous work, our work most closely resembles the Clique Algorithm by [30]. We also use the Incremental Greedy Algorithm originally introduced by Howard et al. in [21]. Additionally, we explored some of the k-connectivity ideas in relation to Potential Fields, such as [22] and [39], although we assume different capabilities which alters how those ideas may be applied. Lastly, we use leader-election and hop-counting to help guarantee connectivity.

Chapter 3

Methodology

3.1 Assumptions

This section describes some of the underlying assumptions that we use in our work. In real environments, these assumptions might not necessarily hold, but they are standard assumptions by other researchers.

1 Wireless Intensity is proportionate to the inverse square of distance.

2 Signal degradation is uniform, with no regard to the presence of walls, other robots, or areas of differing density.

3 Robots can identify the source of a signal by ID.

4 A robot's signal cannot be perceived outside of its communication radius.

3.2 Experimental Design

We utilized Player/Stage [17, 16] robot simulation environment for our experiments along with virtual Pioneer robots. Robotic controllers were implemented in the python programming languages, using the SWIG python interface to the native C Player/Stage libraries.

3.2.1 Experimental Framework

We have created an experimental framework in Python that allows for experiments to be quickly and easily setup and run. Essentially, an experiment creator XML file is run which generates XML files for an experiment, XML files for robots, and the player .world and .cfg files required by Player/Stage to run the robot simulator. While the latter are not complex files, generating these automatically is helpful because Player/Stage does not natively support the dynamic creation of robots. Examples of these files can be found in the Appendix.

3.2.2 Experiment XML file

An experiment XML file consists of two crucial components– a list of variables used in the initialization of the experiment, followed by the list of robots in the experiment itself, which themselves contain both a class reference indicating what type of robot that should be instantiated, and an XML file containing relevant variables that the robot should use to initialize itself. The advantage of this particular approach is that the same code can be used to launch different experiments which each have their own experiment XML file specifying particular experimental conditions. An Experiment file can be found in the appendix.

3.2.3 Robot XML file

This file contains values that should be loaded into the relevant robot class upon startup. The advantage of having each robot's file be separate is that robots can be manually customized if so desired. A Robot file can be found in the Appendix.

3.2.4 Experiment Setup XML file

A single file is used to create all of the above objects needed to run an experiment. Note that an experiment setup XML file may contain more than one type of robot to allow for the creation of heterogeneous systems, although none of our experiments currently use this capability. Thus, by creating (copying and modifying) a simple file, new experimental setups can be created instantly and with ease. An Experiment Setup file with 12 robots can be found in the Appendix.

3.2.5 Analysis Tools

A further advantage of our framework is that a common set of analysis tools can be used to analyze performance across different experiments. The ExperimentAnalyzer.py analysis tool can generate connectivity, coverage, and wireless coverage information, and create screenshots and video of this information. This is limited,

Figure 3.1: Basic structure of object and code relationships in our framework

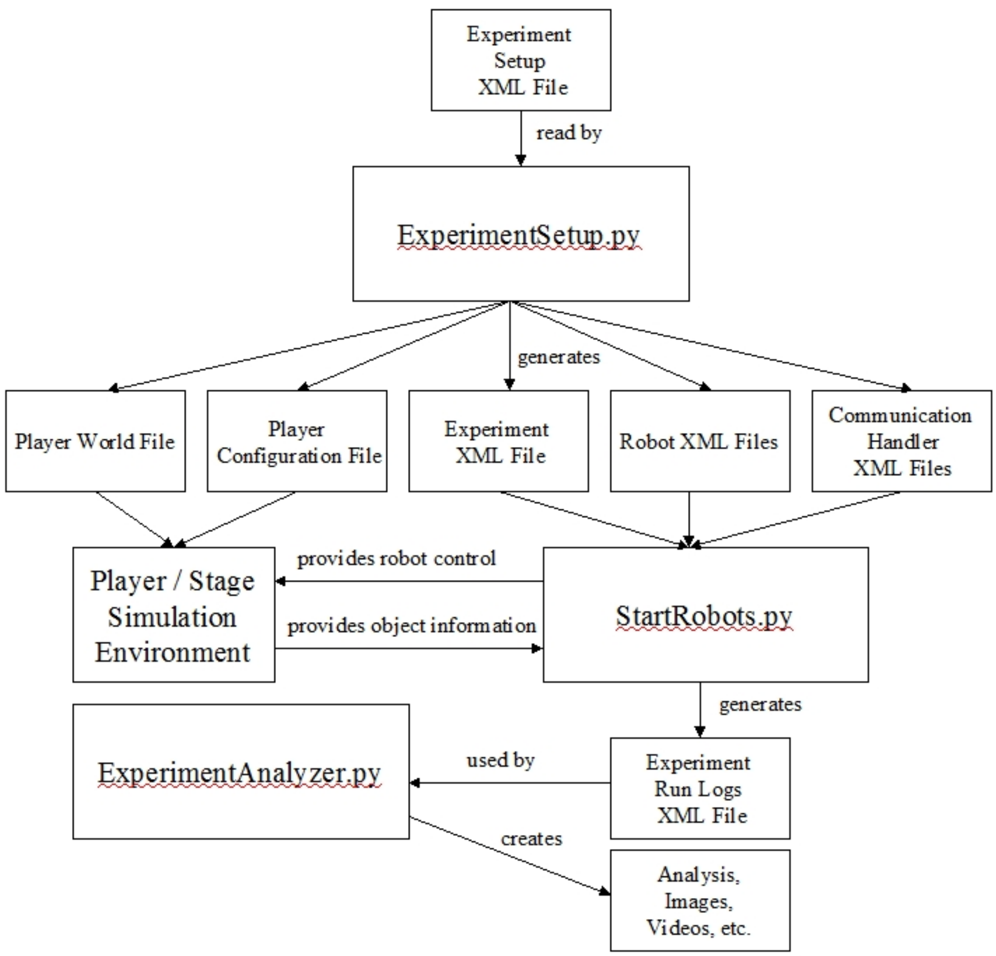
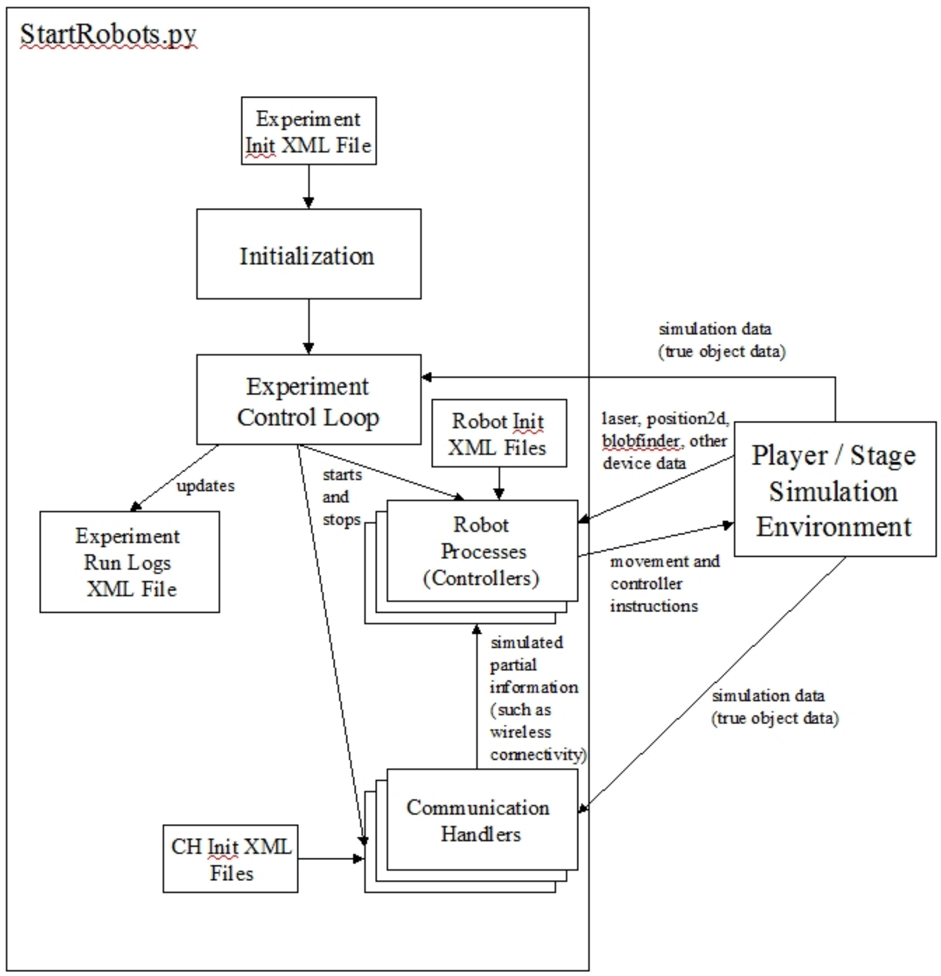


Figure 3.2: Basic structure and flow of an experiment



however, by the sampling interval that is set in the Experiment XML file which adjusts how often robot positions are written to file.

3.2.6 Environments

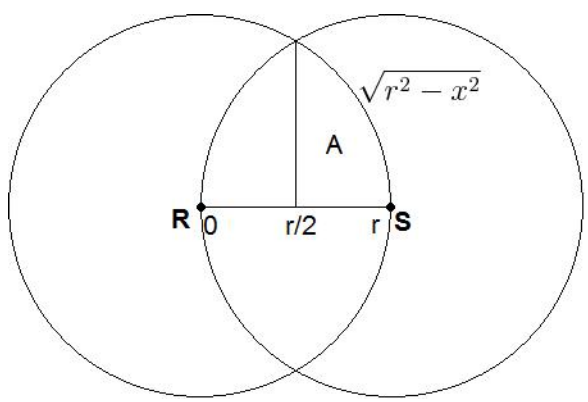
In order to test the generalizability of our approach, it is important to try several different environments. Four different worlds were used: cave, autolab, hospital, and house. The first three are freely distributed with the Player/Stage project. The last is a custom drawing of a house. The four environments vary in complexity with the cave and autolab environments being the easiest, the house environment being slightly more difficult due to the adjoining room, and the hospital environment being the most difficult due to the many rooms in the environment. The starting locations of the robots also have an effect on the complexity of the dispersion problem— some arrangements are easier to disperse from effectively than others.

3.3 Metrics

3.3.1 Coverage

3.3.1.1 Theoretical Maximum Coverage

Figure 3.3: Robots R and S in communication range



A theoretical maximum coverage can be obtained for any number of robots. Consider Figure 3.3. This depicts the minimum overlap of two robots which are in direct communication range with each other. The edges of each of their wireless ranges, depicted by the circles, must include the other robot, which correponds to the center of the other circle. Now assume that each robot has a wireless range with a radius of r meters. The minimum region of overlap between two circles is then $4A = 4\int_{\frac{r}{2}}^{r} \sqrt{r^2 - x^2}\, dx$, which works out to be $\frac{2r^2\pi}{3} - \frac{r^2\sqrt{3}}{2}$. This can be extended more generally; for n robots in a connected network, there must be at least $n-1$ robots mutually in connection range (just as when a connected graph has V nodes, there must be at least V-1 edges). Thus, the minimum possible overlap is necessarily $(n-1)(\frac{2r^2\pi}{3} - \frac{r^2\sqrt{3}}{2})$. It follows then the maximum possible coverage for n robots is $n\pi r^2 - (n-1)(\frac{2r^2\pi}{3} - \frac{r^2\sqrt{3}}{2})$. Please note that this network corresponds to a linear chain of robots. While this would maximize coverage, it is not necessarily feasible in any given environments, nor is it even necessarily desirable. One severe drawback would be the relative brittleness of this network– the loss of any robot besides the two on the ends would be guaranteed to fracture the network into two pieces. Nonetheless, it serves as a tight upper bound on the achievable coverage.

All coverage measurements are in m^2.

3.3.1.2 Incremental Greedy Algorithm

As mentioned earlier, we will be comparing our coverage ratios to those achieved by an implementation of the incremental greedy algorithm outlined by [21]. The basic idea of the algorithm is to sequentially place each robot in such a way that it maximizes the total coverage of the network while maintaining connectivity. Pseudocode for the algorithm follows:

Algorithm 1 Incremental Greedy Algorithm

procedure IncGreedy($\mathbf{R}, x_0, y_0$)
  $\mathbf{R}(1) \leftarrow (x_0, y_0)$
  $\mathbf{C} \leftarrow$ Coverage of R(1)
  $r \leftarrow 2$
  while $r <= |\mathbf{R}|$ **do** ▷ for each robot
    $\mathbf{C}_{best} \leftarrow C$
    Area$_{best} \leftarrow 0$
    for $c = (c_x, c_y) \in \mathbf{C}$ **do**
      $\mathbf{C}_{cur} \leftarrow$ Coverage of R(1), ... ,R(r)
      $Area \leftarrow sum(C_{cur})$
      if $Area > Area_{best}$ **then**
        $\mathbf{C}_{best} \leftarrow C_{cur}$
        $\mathbf{R}_{best} \leftarrow (c_x, c_y)$
      end if
    end for
    $\mathbf{R}(r) \leftarrow \mathbf{R}_{best}$
    $\mathbf{C} \leftarrow \mathbf{C}_{best}$
    $r \leftarrow r + 1$
  end while
  return $\mathbf{R}, \mathbf{C}$ ▷ Robot positions are in $\mathbf{R}$ and coverage is in $\mathbf{C}$
end procedure

It should be noted that the outputs of this algorithm depend greatly on the sampling distance of points to produce $\mathbf{C}$. For example, the spatial configurations of the networks obtained by the incremental greedy algorithm in Figures 3.4, 3.5 and 3.6 are noticeably different, despite the same initial conditions. Specifically, not only do the networks have a somewhat different topology, but even robots in similar locations may have had drastically different trajectories. Presumably, continuing to increase the sample rate would eventually yield limiting behavior, but since increasing the sampling rate by a factor of n increases the algorithm run time by n^2, this limit is expensive to find. It also should be noted that since the algorithm is greedy, an initially profitable location may be preferred early which might ultimately lead to a loss of overall coverage. Thus, increasing the sampling rate is not guaranteed to increase the final coverage. Additionally, given the fact that the network found by the incremental greedy algorithm is almost certainly suboptimal to begin with, we did not increase our sampling rate beyond .08 due to time constraints. This took about 8 hours on a 3.6GHz computer to deploy 24 robots. This was with further optimizations which reduce repeated sampling by maintaining a memory for each point and exploiting the fact that the coverage increase of adding a robot there is nonincreasing with the number of robots. Once again, our goal is primarily to achieve a lower bound for a semi-realistic optimal placement and provide a common metric of reference for our results. Table 3.1 contains the coverage values obtained after the placement of each successive robot.

Figure 3.4: Cave– Incremental Greedy Algorithm, .32 sample distance: The numbers detail the order in which the network was deployed.

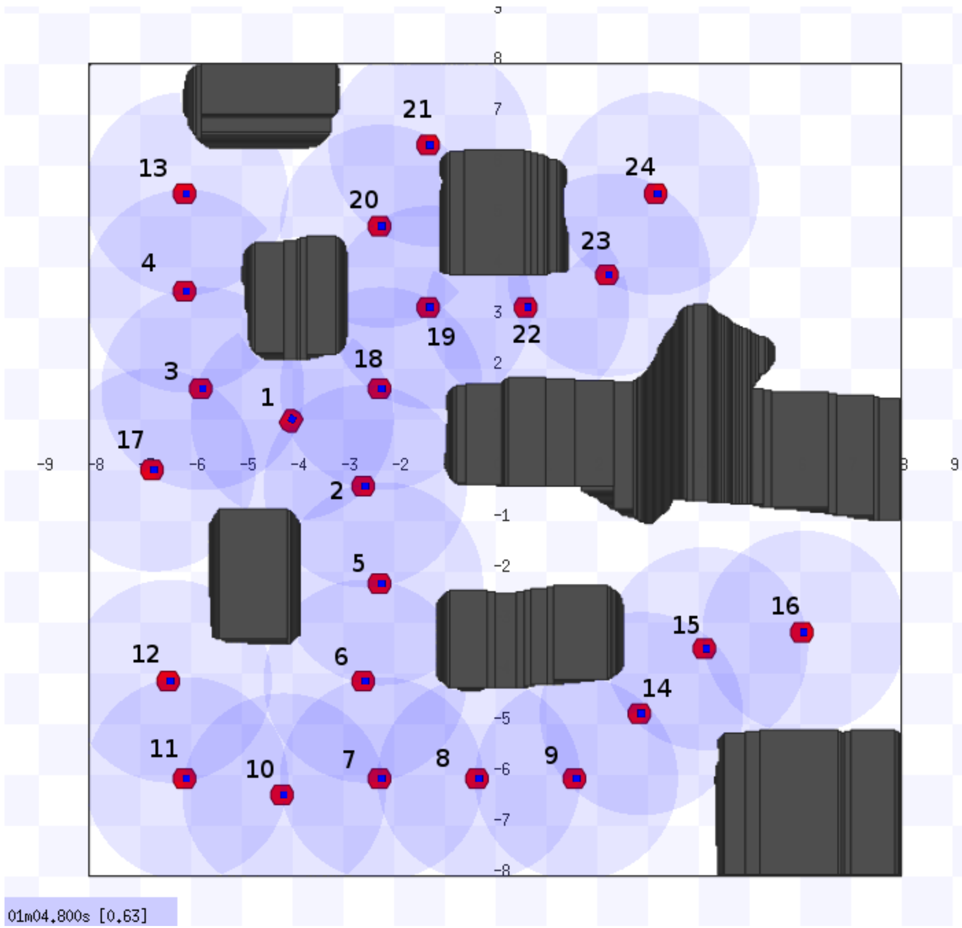


Figure 3.5: Cave– Incremental Greedy Algorithm, .16 sample distance, same initial conditions as in 3.4

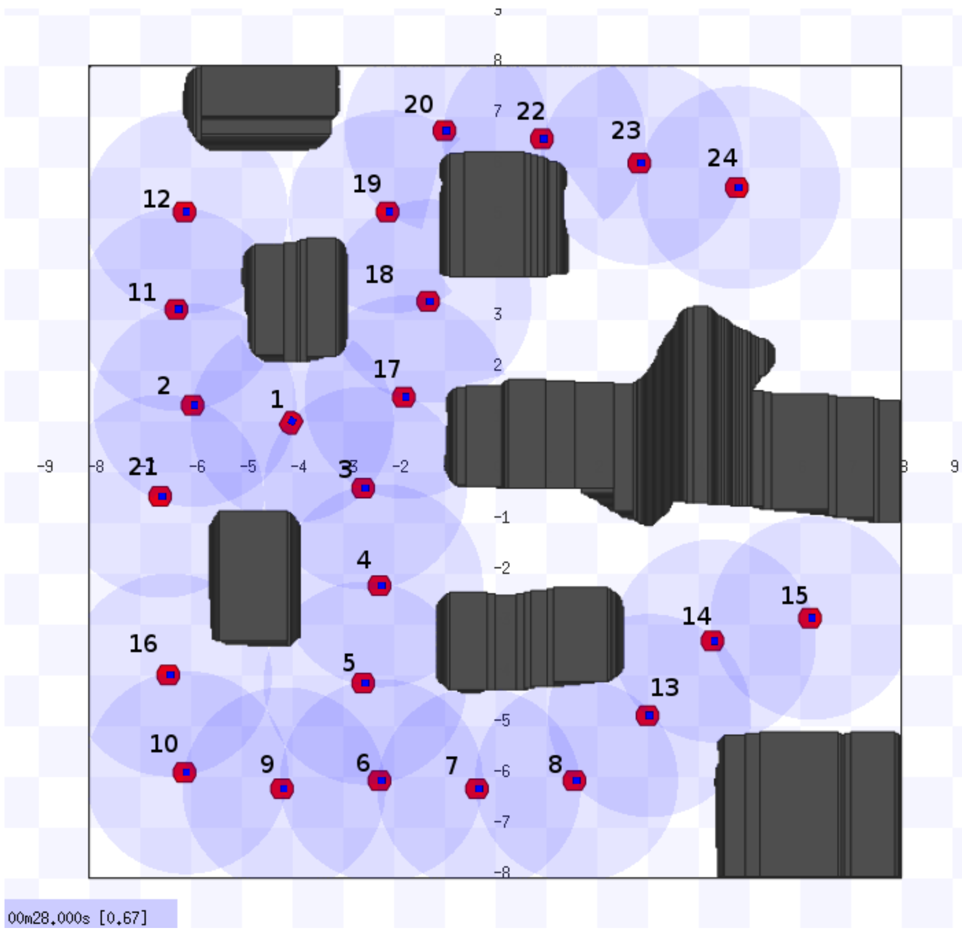


Figure 3.6: Cave– Incremental Greedy Algorithm, .08 sample distance, same initial conditions as in 3.4

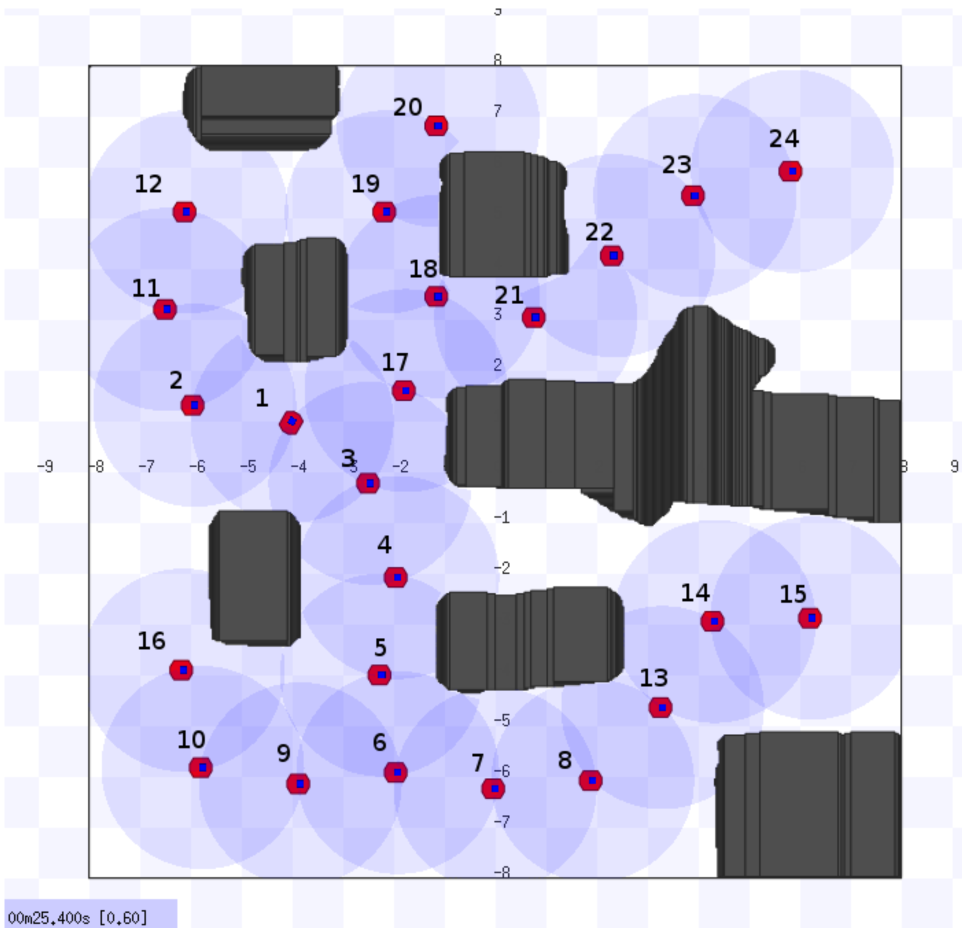


Table 3.1: Cave– Incremental Greedy Algorithm, Coverage, 1-24 robots, various sampling rates

Robots Deployed	**.32 Sample Rate (m^2)**	**.16 Sample Rate (m^2)**	**.08 sample Rate (m^2)**	**Theoretical Maximum (m^2)**
1	10.2912	10.2912	10.2912	12.5664
2	17.4490	17.6169	17.6169	23.2195
3	24.3087	24.7747	24.8474	33.8727
4	30.3616	30.6606	30.8378	44.5258
5	36.2476	37.3965	37.3156	55.1790
6	42.9834	44.5819	44.5573	65.8321
7	50.1688	51.2451	51.3761	76.4853
8	56.7992	58.2810	58.4468	87.1384
9	63.8188	64.4649	64.8888	97.7916
10	69.9259	71.6534	72.1326	108.4450
11	77.0621	77.5260	78.0933	119.098
12	82.9399	83.7765	84.3469	129.751
13	88.8125	89.4925	90.2420	140.404
14	94.0892	96.5970	97.4592	151.057
15	100.6428	103.5858	104.2924	161.71
16	107.7545	109.2966	110.1169	172.364
17	112.6932	114.3163	115.3167	183.017
18	117.4211	119.4445	120.2688	193.67
19	122.5943	125.0294	125.9530	204.323
20	127.5943	129.7132	130.6778	214.976
21	132.6602	134.2095	135.2499	225.629
22	136.9498	138.6394	141.9725	236.282
23	143.1634	146.2917	149.2603	246.936
24	149.9924	153.6287	156.5972	257.589

Figures 3.7, 3.8, and 3.9 show the results of the Incremental Greedy algorithm on a few different environments. Table 3.2 contains the coverage areas obtained after the placement of each successive robot. Note that although the values are not directly comparable due to differences in both the initial robot placement and the environment, the more constricted and complex the environment, the lower the final coverage obtained. This makes sense as there are more walls and barriers occluding some or all of the robots' vision. One other thing to note is that frequently as the complexity of the environment increases, the solution obtained by the incremental greedy algorithm is somewhat impractical. This is because the incremental greedy algorithm has no problem skipping over walls and other barriers even if the distance a robot would have to navigate to get past the wall would far exceed the simple Euclidean distance between the points. This makes it difficult for real-world approaches to replicate the coverage results obtained by this method. Again though, it is useful as a comparison since it provides a lower bound for the best theoretical possible coverage obtainable. Further note that due to the incremental greedy nature, networks of up to 24 robots using the same initial conditions can be constructed from this data. For example, an incremental greedy experiment with twelve robots using any environment would use the robots labeled one through twelve on that same environment.

Figure 3.7: House– Incremental Greedy Algorithm, .08 sample distance

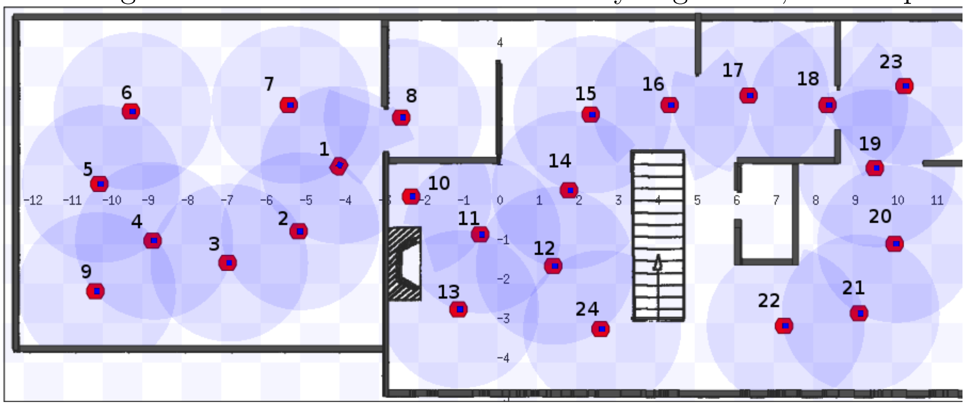


Figure 3.8: Hospital Section– Incremental Greedy Algorithm, .08 sample distance

Figure 3.9: Autolab– Incremental Greedy Algorithm, .08 sample distance

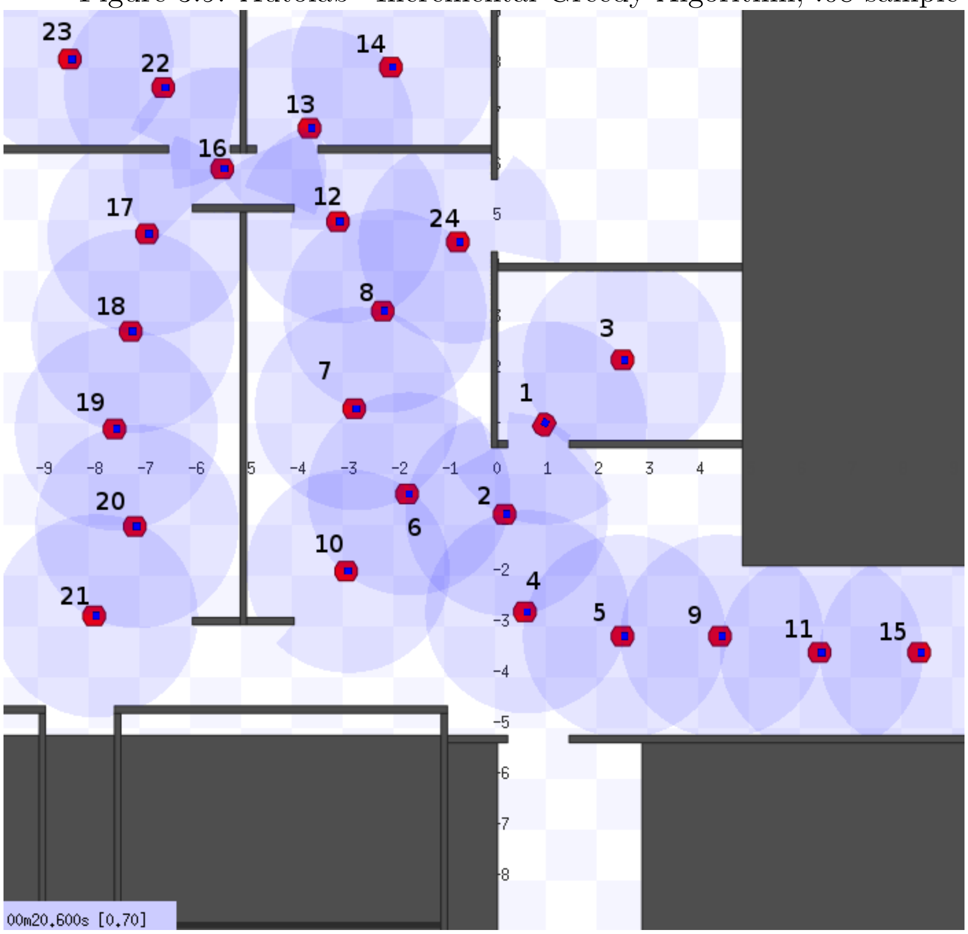


Table 3.2: Cave, House, Autolab, Hospital– Incremental Greedy Algorithm Coverage, 1-24 robots, 0.08 sample rate

Robots **Deployed**	**Cave** (m^2)	**House** (m^2)	**Autolab**	**Hospital Section** (m^2)	**Theoretical Maximum** (m^2)
1	10.2912	9.7643	10.6753	7.0579	12.5664
2	17.6169	17.8150	18.2059	14.6156	23.2195
3	24.8474	25.3255	25.2752	21.47593	33.8727
4	30.8378	32.8810	32.4255	28.2928	44.5258
5	37.3156	40.3323	39.4962	34.8672	55.179
6	44.5573	47.3282	46.6479	41.0167	65.8321
7	51.3761	54.2803	53.7602	47.2080	76.4853
8	58.4468	60.0168	60.3271	52.9264	87.1384
9	64.8888	65.2192	66.4276	60.3943	97.7916
10	72.1326	70.3191	72.8459	66.4630	108.445
11	78.0933	74.5599	79.2263	73.0584	119.098
12	84.3469	82.0436	85.0698	80.5158	129.751
13	90.2420	89.6622	90.6744	86.1669	140.404
14	97.4592	95.2548	97.0102	92.9299	151.057
15	104.2924	101.4680	104.2460	99.1886	161.71
16	110.1169	107.9200	110.8717	104.6676	172.364
17	115.3167	115.2280	115.2061	110.0328	183.017
18	120.2688	120.2470	122.3306	115.2065	193.67
19	125.9530	126.4086	127.9187	120.2065	204.323
20	130.6778	132.6023	135.4565	125.2201	214.976
21	135.2499	140.3565	142.4992	130.3294	225.629
22	141.9725	147.8831	149.5397	136.8634	236.282
23	149.2603	153.6601	156.6226	142.0206	246.936
24	156.5972	160.2728	163.5878	146.7588	257.589

3.3.2 Connectivity

Another important aspect of a deployment is its connectivity. There are a few important metrics here; we are interested in the size of the largest connected graph and its coverage, and the effects of node failure, among others. Connectivity is also a major concern during the process of deploying the network. If a robot or robot cluster manages to somehow become isolated, it may never reconnect with the rest of the network. This naturally significantly reduces the possible maximum coverage that one can achieve and it is something that we obviously want to avoid.

Algorithm 2 contains the algorithm we used for determining the connectivity of a network given an adjacency matrix representing whether a pair of robots are connected and the number of robots. This algorithm begins with all vertices each in their own group. At every step, groups that communicate with each other directly are merged. Similarly, groups that display transitive connections are merged.

Algorithm 2 Connectivity Algorithm

```
procedure CONNECTIVITY(X, n)    ▷ X is a nxn matrix and n is the # of robots. X(i,j) is marked 1 if robot i can contact robot j, and 0 otherwise.
    for i ∈ [1, 2, ..., n] do    ▷ L is a nx1 matrix with each index containing lists of a single number initially
        L(i) ← {i}
    end for
    nc ← 0
    while nc < n do
        for i ∈ [1, 2, ..., n]  do
            MERGE = ∅
            PMERGE = ∅
            for j ∈ [i + 1, ...n] do
                if X(i, j) = 1 and X(j, i) = 1 then
                    MERGE ← MERGE ∪ {j}
                end if
                if X(i, j) = 1 and X(j, i) = 0 then
                    PMERGE ← PMERGE ∪ {j}
                end if
            end for
            for t ∈ MERGE do    ▷ Any two communicating groups merged.
                for y ∈ [1, 2, ..., n] do
                    X(i, y) ← X(i, y)|X(t, y)
                    X(t, y) ← 0
                end for
                for y ∈ [1, 2, ..., n] {i} do
                    if X(y, t) = 1 then
                        X(y, t) ← 0
                        X(y, i) ← 1
                    end if
                end for
                L(i) ← L(i) ∪ L(t)
                L(t) ← ∅
            end for
            for t′ ∈ PMERGE do    ▷ Transitivity to resolve cycles.
                for y ∈ [1, 2, ..., n] do
                    X(i, y) ← X(i, y)|X(t′, y)
                end for
            end for
        end for
        nc ← nc + 1
    end while
    return X, L    ▷ Connectivity between subgroups in X and subgroups in L
end procedure
```

3.3.3 Dispersion Rate

A third metric for evaluating dispersion algorithms is the dispersion rate. Naturally the deployment speed of algorithms is a critical component in their usefulness– an algorithm which took a very long time to deploy a perfect network would not be universally useful. There are many applications such as military, emergency response, or intruder alert, where a quick, suboptimal spread of nodes is more desirable than a slower, higher coverage approach.

3.3.4 Cost & Power Usage

One metric that can be used to measure power usage directly is measuring the total distance traveled by each robot. Another important indicator is how many messages were sent. While extremely important to any practical robot swarm deployment, we ignore this particular dimension for the purposes of our experiments.

Chapter 4

Preliminary Experiments

This section will outline algorithmic approaches and some of our findings from preliminary experiments.

4.1 Potential Fields

Potential fields, or attract/repel approaches, as an approach to dispersion lends itself to a few different problems. The first issue that we discovered in our experiments was that the runs were very sensitive to minor random fluctuations. Despite the initial conditions being the same for each run, the final deployed networks were very different in both positioning for each robot, and frequently in the overall pattern of dispersion at the end of the experimental period. This does not necessarily mean that the overall coverage and connectivity drastically varied from run to run, but rather that the network in existence in the end was qualitatively different. Even removing sources of randomness does not eliminate this phenomenon in simulation, due to unpredictable multithreading effects. It probably also can be extrapolated to a real deployment– perfect synchronization in a system is difficult, and in a noisy world, messages will surely be lost on occasion.

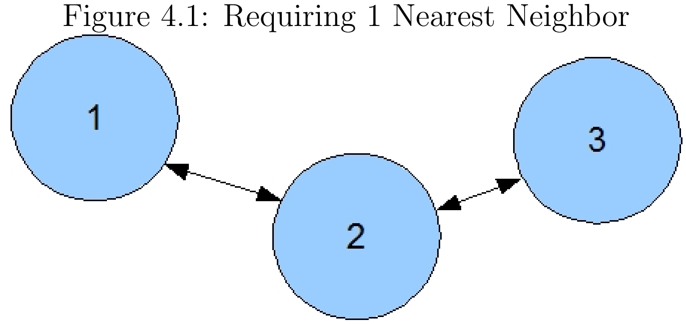
Another issue with the potential field approach is that it is possible for robots to become stuck in suboptimal positions due to the placement of terrain and other robots. This can cause the entire system to also become stuck in a suboptimal configuration. Much less frequent, but still possible is the occurrence of periodic oscillations in robot actions.

One last characteristic of potential fields is that a small fluctuation in one robot's position can cause other robots to adjust their own positions and this can ripple through the entire network. We found by trial and error that incorporating a 'dead-zone' in which a robot is less or non-responsive to changes helps limit these effects since minor changes in a neighbor's position will not necessarily force a reassessment, which in turn damps the effects of minor perturbations on the network as a whole.

4.2 Nearest Neighbor Algorithms

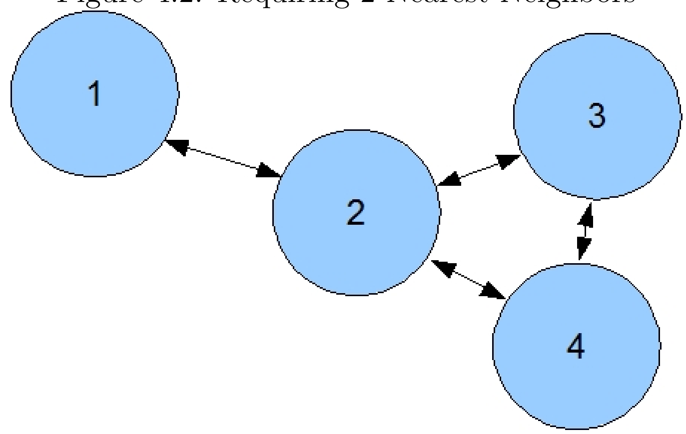
This designation characterizes a class of distributed algorithms which rely on each robot keeping one or more neighbors in range in order to move. This class of algorithms suffers from serious connectivity issues in practice. Figures 4.1, 4.2, and 4.3 demonstrate a few of the problems.

Figure 4.1: Requiring 1 Nearest Neighbor



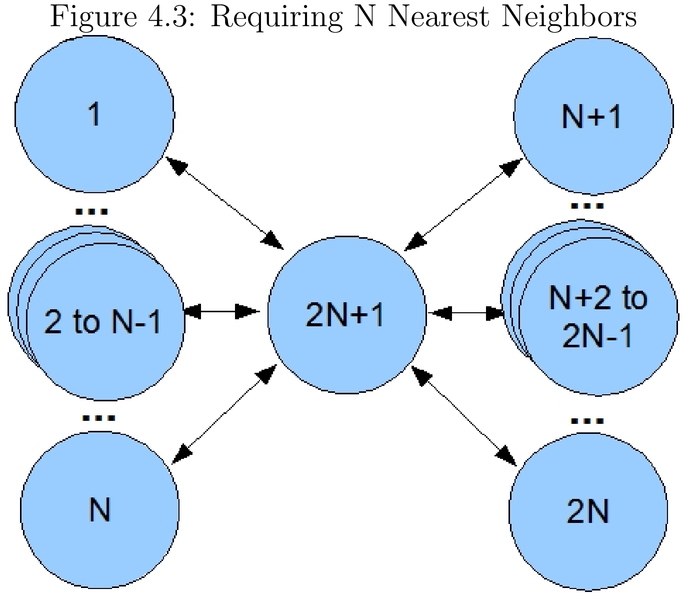
Robots 1 and 3 each share one neighbor, Robot 2. They also have no other neighbors. Thus, they are fixed. On the other hand Robot 2 has two neighbors and may wander. In the process of doing so, it may disconnect either Robot 1 or Robot 3.

Figure 4.2: Requiring 2 Nearest Neighbors



Robots 1 and 4 each begin with exactly two neighbors, and will not move, whereas Robots 2 and 3 each have 3 neighbors. If Robots 2 and 3 move towards Robot 1 for instance, the network connection with Robot 4 will be severed and the network is now disconnected.

Figure 4.3: Requiring N Nearest Neighbors



In the diagram below, Robot 2N+1 can go right or left and either way destroy the connectivity of the network. Admittedly this is a pathological and relatively unlikely case.

As these figures demonstrate, N-nearest neighbor algorithmic approaches to dispersion can always theoretically fissure into two or more disconnected networks. Even in the the most general case of N-nearest neighbors, a group of N neighbors can form a group that can cause the network to become disconnected. In particular, it is possible in a pathological arrangement, such as in in Figure 4.3, would allow for one robot to be simultaneously connected to two groups of N robots and move in such a way that it would disconnect the network. Granted, with more neighbors, the chance of this happening in practice drops dramatically. Nonetheless, the approach incurs an extremely heavy cost as there will necessarily be much overlap in coverage since each robot will need to maintain N neighbors. Optimal single-coverage schemes will seek to minimize this– each robot should ideally cover as much unique area as possible. As the required connectivity, N, increases, the total coverage of the system will necessarily drop as the entire network will need to be more closely clustered together in order to satisfy the N-nearest neighbor requirement.

4.3 Counting Hops

This algorithmic approach should be characterized as semi-centralized and involves the additional cost of communication. There are two essential phases in this approach; the first phase has the robots select a common reference point, or leader, and the second phase involves using the leader as a common reference point to coordinate connectivity and movement.

The first part mentioned above, we will largely ignore– leader election algorithms being beyond the scope of this work. However, if we make the not unreasonable assumption that each robot was assigned a unique ID, we note that if nothing else, a O(r) algorithm is sufficient to find the robot with the lowest ID, where r is the number of robots. This would be achieved by each robot broadcasting the minimum of its own ID and the lowest ID that it has ever received. After r iterations, the entire (connected) network will have the same ID as the lowest, and this would become the leader.

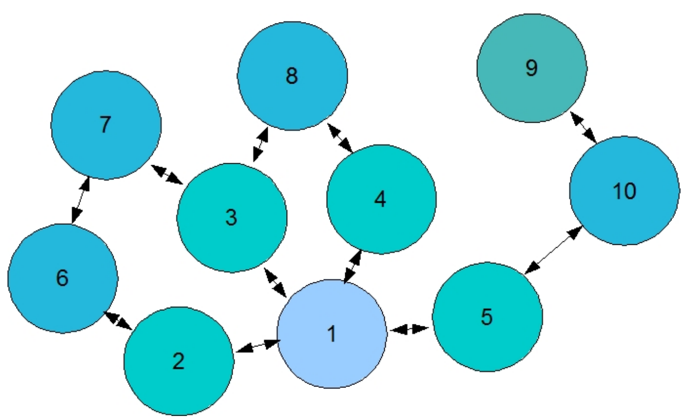
Once the leader is established, robots will count hops to the leader. A neighbor hop count of zero is assigned to the leader itself. Other robots calculate their own hop count by measuring broadcasts from neighbors and assigning themselves a hop count corresponding to the lowest neighbor hop count plus one. This can be seen in equation 4.1, where H_R is the hop count of robot R, and N_R is the neighbors of robot R.

$$H_R = \min_{N \in N_R} (H_N) + 1 \tag{4.1}$$

The basic idea is that while moving, each robot should maintain at least one neighbor with a lower hop count. The leader does not move and provides an anchor for the entire network. This helps to further the goal of reducing disconnections in the network. Unfortunately this approach is not without its own flaws; it is still possible for robots to drift apart, severing the network. Figure 4.4 demonstrates the counting hops idea and how it works, and 4.5 illustrates the fundamental flaws.

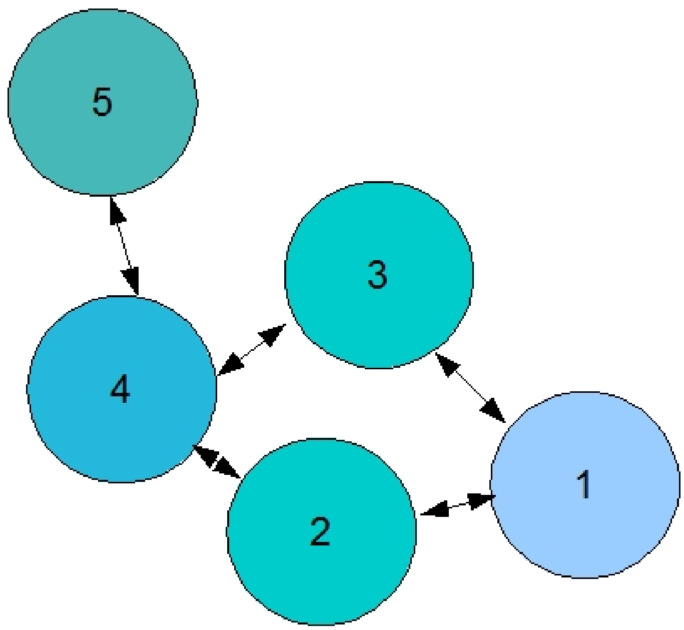
This scheme improves upon the topology based movement described earlier because it's easy for a robot to query its neighbors, find out how close in hops they are to the leader relative to their neighbors, and use this information to move. In our trials, this improved both spread and connectivity quite a bit, but it does introduce new drawbacks as mentioned. These are communication cost and the introduction of a single point of failure. Recovery is theoretically possible, via a new leader election, but there is no guarantee that the resulting network will still be connected. It is possible to combine counting hops and leader

Figure 4.4: Counting Hops



The leader in this diagram is robot 1. Robots 2, 3, 4, and 5 are one hop away. Robots 6, 7, 8, and 10 are two hops away. Robot 9 is three hops away.

Figure 4.5: Counting Hops– Problems



Counting hops is not by itself a panacea; it is still possible for robots to drift apart. Consider the above diagram. Robot 4 has two neighbors (2 and 3) with a smaller hop count so it considers itself free to wander. Robot 5 in contrast only has Robot 4 as a neighbor with a higher hop count, and therefore stays put. Robot 4 may in its wanderings disconnect Robot 5, disconnecting the network.

4.4 Alarm Counting Hops

This approach is similar to the counting hops idea, except that it also incorporates some ideas similar to the Clique approach by [30]. In the Clique Algorithm, the entire network was analyzed and certain robots were selected as sentries which could not move. This approach uses a similar sentry idea to restrict the ability of robots to move and prevent the network from fissuring as could happen above. It uses explicit rather than implicit communication in order to accomplish this– robots may now send alarms to freeze certain neighbors. A robot sends an alarm if and only if there are no robots with a lower hop count with a signal threshold of α where this threshold is defined as .8 of the maximum communication range. An alarm will have a target– the target of this alarm must immediately stop what it is doing and freeze. This helps to enforce the condition that robots stay together and in practical terms prevented the network from breaking apart where the other approaches did not. This simple basic approach and its derivatives were what we ended up using in our experimental versions.

Chapter 5

Experiments

5.1 Initial Setup

The robots were placed in a very tightly packed area and in such a way so that the network is connected. The connectedness is a requirement for this particular algorithm to work, as it does not inherently contain any method for repair, except that a robot can reconnect via random wandering. We do not feel that this a flaw of the algorithm, however; any approach that can handle a disconnected network in a consistent fashion would require more powerful hardware than we are assuming. For the first series of experiments, we place each robot in the same place with the same facing. The initial setup of the robots for each experiment can be found in Figure 5.1, Figure 5.2, Figure 5.3, and Figure 5.4.

Figure 5.1: Cave starting positions

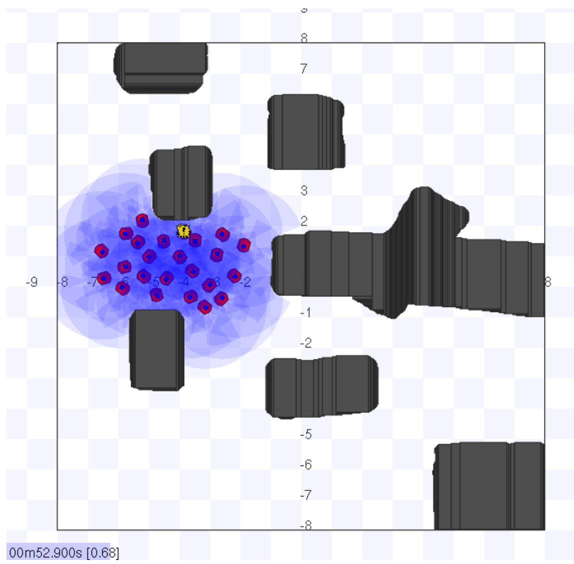


Figure 5.2: House starting positions

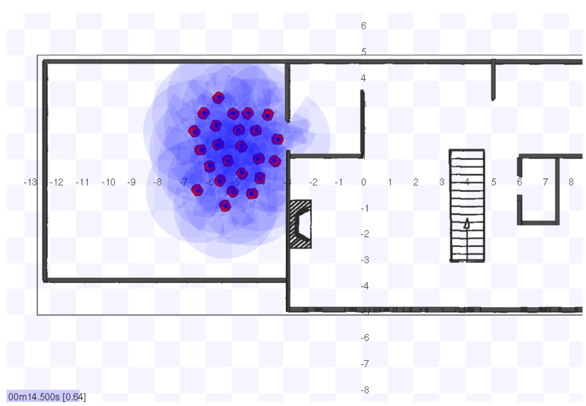


Figure 5.3: Autolab starting positions

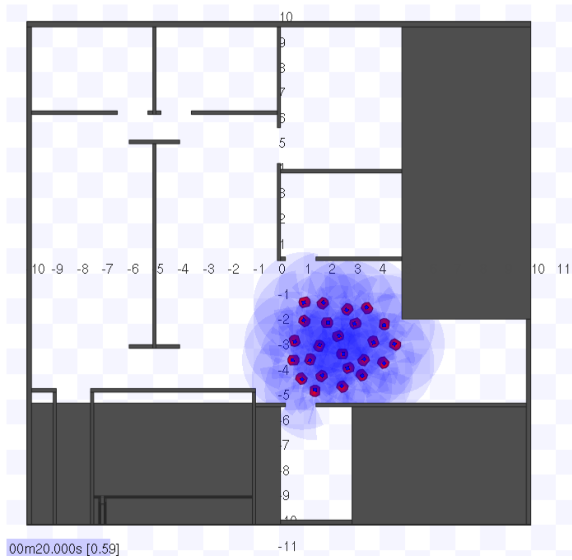
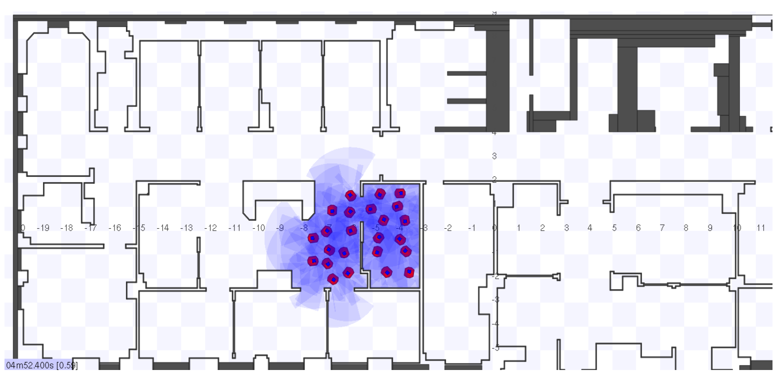


Figure 5.4: Hospital starting positions



5.2 Experimental Parameters

5.2.1 Parameters

The specific parameters used in our experiments are listed in Table 5.1.

Parameter	**Description / Value**
radius	2 (wireless & sight)
μ	total # of robots within communication range
λ	10
F_R	Radial Force
F_T	Tangent Force
F_O	Open Force

Table 5.1: Parameters

5.2.2 Potential Fields Setup

We have three main potential field forces. The first, and largest, is a radial force whose magnitude is determined by distance. We use a combination of linear and exponential magnitudes to determine the force. The exact equation can be seen in equation 5.2. Here, **p** is a laser measurement corresponding to a point mass (obstacle) along a vector. d(**p**) refers to the distance of that point mass and m(**p**) is the resulting force exerted on the robot by that measurement. I corresponds to an indicator function which evaluates as 1 if its condition is true and 0 otherwise.

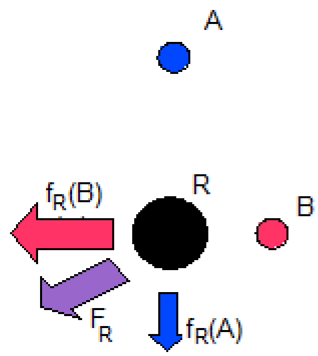
$$m(\mathbf{p}) = 4/d(\mathbf{p}) + I_{d(\mathbf{p})\leq 1}e^{-d(\mathbf{p})}) \tag{5.1}$$

$$F_R = \sum_{\mathbf{p}\in P} m(\mathbf{p}) \parallel \mathbf{p} \parallel \tag{5.2}$$

Figure 5.5 shows the effect of two distance readings A and B upon a robot R.

There is an additional radial force generated by open areas that pull robots in the direction of empty regions. We call this the open force, F_O. It is strongest in large contiguous regions of high distance readings, indicating a relatively open area. It differs from the Radial Force in that the Radial Force pushes robots away from point masses, whereas the Open Force pulls them towards regions in their view without point masses.

Figure 5.5: Radial Force, f_R, purple arrow, determined by two point masses A and B. The forces that A and B exert upon R are shown by the red and blue arrows, respectively.



Finally, there is a tangential force generated from each pair of successive point readings. The underlying assumption is that the geometry between successive readings is a straight line, representing a wall or another obstacle. It is important to note that this assumption does not always hold– but rather relies on the conception of the environment as being smooth. The line between two points itself offers two different candidates for a tangential force. This ambiguity is broken by using the radial force, F_R. Specifically, the candidate with a smaller angle relative to the Radial force vector is used. The magnitude of the tangential force is determined by the closer of the two points, but is smaller than the radial force.

The tangent force generated from a pair of readings can be seen in equation 5.3.

$$f_{a,b} = \| \min_{\mathbf{n} \in \{(\mathbf{a}-\mathbf{b}),(\mathbf{b}-\mathbf{a})\}} |\arccos(F_R) - \arccos(\mathbf{n})| \, \| \, \frac{\max(m(a), m(b))}{4} \tag{5.3}$$

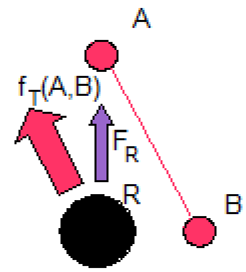
The total tangential force is simply this force calculated over every pair of successive readings, p and p'. This is in Equation 5.4.

$$F_T = \sum_{\{p,p'\} \in P} (f_{p,p'}) \tag{5.4}$$

Figure 5.6 shows an example of how the tangent force is calculated.

These two forces combine to push robots around. However, these approaches offer no innate support for preserving connectivity between robots.

Figure 5.6: Tangent Force, f_T, purple arrow, determined by two point masses A and B.



5.2.3 Robot Behaviors

Rodney Brooks, in his classical work [10], demonstrated the usefulness of modular behaviors in mobile robotics with his subsumption architecture. In [4], Arkin further expounded the benefits of a multi-layered behavioral approach. The key points of this work were that (1) multiple behaviors can provide a modular, distributed framework, (2) behaviors may process different input and be active at different times, and (3) behaviors can be layered to obtain complex interactions to tackle a problem. As many others before us in the mobile robotics community, we will be using these basic ideas as inspiration for our robot architecture.

In the Alarm Counting Hops algorithm, each robot has six different behaviors, namely Random, Backup, Forward, RandomTurn, Freeze, and Scatter. The initial behavior for all robots is Scatter. When a collision occurs, or a robot gets stuck, the robot may randomly pick another behavior. It then pursues that for a few seconds, before picking yet another new behavior. This process continues until the robot is once again in the Scatter state. The exception is the Freeze state which instructs the robot to cease moving altogether in order to prevent network disconnection. The leader (in our experiments the robot with ID 1) stays in Freeze and does not move, thereby providing an anchor for the network.

5.2.3.1 Random

The Random behavior allows robots to pick a random turn velocity and forward velocity.

5.2.3.2 RandomTurn

The RandomTurn behavior allows robots to reorient themselves in place. The robot picks a random direction to rotate in and does so for a few seconds with no forward velocity. This behavior gives way to any of the other behaviors after 5 seconds.

5.2.3.3 Scatter

This is the primary behavior in which robots spend most of their time, only reverting to other behaviors upon collision. As we have discussed earlier, nearby obstacles provide two forces: the radial force repelling the robot and a tangential force that causes the robot to follow a 'wall'. Finally, the open force acts upon the robot, attracting it towards large open gaps.. These vectors are summed and normalized and determine the robot's direction and velocity.

5.2.3.4 Forward

The Forward behavior is identical to the Scatter behavior above, except the resultant vector also sums a strong positive vector in the current direction the robot is facing. This has the effect of influencing the robot to continue moving in the direction it is facing.

5.2.3.5 Backup

The Backup behavior is identical to the Scatter behavior above, except the resultant vector also sums a strong vector in the opposite direction of the current direction the robot is facing.

5.2.3.6 Freeze

This behavior may be triggered when movement may cause the network to be disconnected, namely when either a robot is sending an alarm, or another robot is sending an alarm to it.

5.2.4 Robot Decision Flow

Algorithm 3 Alarm Counting Hops – Robot State Decision Flow

```
procedure RobotDecisionLoop
    if robot has ≥ 1 neighbor broadcasting an alarm signal then
        State ← Freeze
    else
        if State == Freeze and robot has > 1 neighbor with lower hops then
            State ← Scatter
        else
            if robot has ≤ 1 neighbor with a lower hop count then
                robot sends alarm to neighbor with lowest hop count, then ID.
                State ← Freeze
            else
                if robot is receiving an alarm then
                    State ← Freeze
                else
                    if robot is stuck then
                        20% chance State ← Backup
                        20% chance State ← RandomTurn
                        20% chance State ← Forward
                        20% chance State ← Random
                        20% chance State ← Scatter
                    end if
                end if
            end if
        end if
    end if
end procedure
```

5.2.5 Robot Execution Flow

Algorithm 4 Alarm Counting Hops – Robot Execution Flow

procedure RobotExecution
    if State == Freeze **then**
        robot does not move
        break
    end if
    if State == RandomTurn **then**
        $\theta \leftarrow \textbf{random}(0, 2\pi)$
        robot turns towards θ
        robot speed determined by closest point mass
        break
    end if
    if State == Scatter **then**
        $\mathbf{v} \leftarrow \{0, 0\}$
    end if
    if State == Random **then**
        $\theta \leftarrow \textbf{random}(0, 2\pi)$
        $\mathbf{v} \leftarrow \{\lambda \cos\theta, \lambda \sin\theta\}$
    end if
    if State == Forward **then**
        $\theta \leftarrow$ **current facing**
        $\mathbf{v} \leftarrow \{\lambda \cos\theta, \lambda \sin\theta\}$
    end if
    if State == Backup **then**
        $\theta \leftarrow$ **current facing**
        $\mathbf{v} \leftarrow \{\lambda \cos\theta, \lambda \sin\theta\}$
    end if
    $\mathbf{v} \leftarrow \mathbf{v} + F_R$ ▷ Add Radial Force (Figure 5.5)
    $\mathbf{v} \leftarrow \mathbf{v} + F_T$ ▷ Add Tangent Force (Figure 5.6)
    $\mathbf{v} \leftarrow \mathbf{v} + F_O$ ▷ Add Open Force
    robot turns towards $\arctan v.y/v.x$
    robot speed determined by closest point mass
end procedure

5.2.6 Communication

We use the Alarm Counting Hops algorithm introduced in the last section. We assume that each robot has a unique ID and also that the robot with ID 1 has already been elected leader. As mentioned earlier, there exists at least one known algorithm to do leader election, so this assumption seems reasonable, as leader election is not the main focus of this work. Robots may send an alarm if they have only a single neighbor with a lower hop count and that neighbor is sufficiently far away. Alarm messages are directed and instruct a specific robot to cease movement, which it does immediately upon reception.

5.2.7 Experiments

There were three broad set of experiments that we did in this paper. The first was to measure sensitivity differences caused by minor changes in robot position. To this end, we compared the results of 20 fixed initial positions against 20 randomized initial positions on the cave environment. The next set of tests was to measure the performance of the approach on different environments: twenty runs of each of the cave, hospital, house, and autolab environments mentioned earlier were all performed. The cave and autolab environments were the more forgiving testbeds, while the house and especially the hospital proved more difficult because robots had to leave or enter rooms to maximize coverage. The last set of experiments was measuring the effects of reducing the number of robot laser readings on the cave environment in an effort to see how many readings were necessary to achieve good performance and thus to see how primitive the robots could be.

The first experiment we ran was with twenty-four robots in the Cave Environment. The initial setup can be seen in Figure 5.1. The results in the tables below are each based on a set of twenty runs. Readings were not always measured at the same time; so linear interpolation was used to estimate coverage every twenty seconds.

Two different types of experiments were run on the cave environment. The very first was with fixed start positions for the robots. The average of twenty runs can be seen in Table 5.2. Coverage increases from the start to about an average of $81.443m^2$ coverage from a starting coverage of $35.372m^2$. After 300 seconds, the robots covered $74.002m^2$, and 600 seconds, $78.182m^2$. Thus, at least with this particular setup, roughly 83.85% of the expansion is done by 300 seconds, and by 600 seconds, roughly 92.92% of the coverage is completed. Connectivity is generally maintained throughout, however, it was still possible for the algorithm to disconnect at times.

The final total fell well short of the lower bound on maximum coverage predicted by the incremental coverage algorithm, obtaining $81.433m^2$ / $156.5972m^2$ in the end, or 52.00% of the possible expansion. Again though, the incremental algorithm is not actually possible in a real deployment. Additionally, such a deployment is optimized to maximize coverage and would not be as robust to failures.

Figure 5.7: Fixed Cave, Coverage over Time

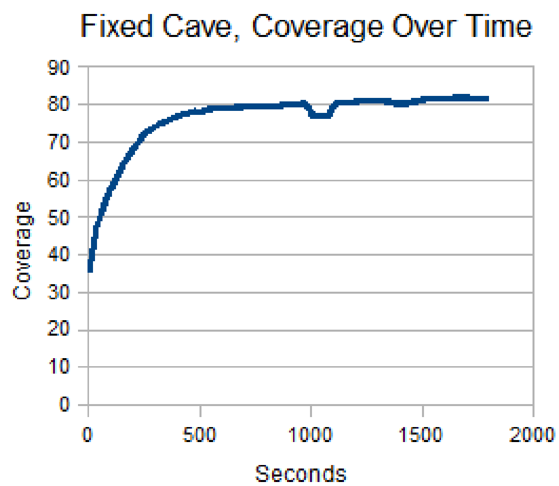


Table 5.2: Cave, Fixed Positions

Time	**Cov.** (m^2)	**Con.**	**Time**	**Cov.** (m^2)	**Con.**	**Time**	**Cov.** (m^2)	**Con.**
0	35.372	24.0	620	78.818	24.0	1220	80.631	24.0
20	41.359	24.0	640	78.838	24.0	1240	80.758	24.0
40	47.345	24.0	660	78.916	24.0	1260	80.786	24.0
60	51.752	24.0	680	79.007	24.0	1280	80.778	24.0
80	54.775	24.0	700	79.094	24.0	1300	80.757	24.0
100	57.352	24.0	720	79.193	24.0	1320	80.802	24.0
120	59.719	24.0	740	79.254	24.0	1340	80.863	24.0
140	61.956	24.0	760	79.254	24.0	1360	80.917	24.0
160	64.199	24.0	780	79.234	24.0	1380	80.591	23.667
180	66.233	24.0	800	79.246	24.0	1400	80.107	23.667
200	68.004	24.0	820	79.291	24.0	1420	80.176	23.667
220	69.671	24.0	840	79.334	24.0	1440	80.143	23.667
240	71.325	24.0	860	79.433	24.0	1460	80.136	23.667
260	72.539	24.0	880	79.532	24.0	1480	80.574	23.667
280	73.161	23.857	900	79.646	24.0	1500	80.962	24.0
300	74.002	23.857	920	79.777	24.0	1520	81.065	24.0
320	74.952	24.0	940	79.884	24.0	1540	81.185	24.0
340	75.428	24.0	960	79.932	24.0	1560	81.296	24.0
360	75.845	24.0	980	79.996	24.0	1580	81.379	24.0
380	76.236	24.0	1000	80.039	24.0	1600	81.403	24.0
400	76.646	24.0	1020	77.88	22.857	1620	81.424	24.0
420	77.079	24.0	1040	76.739	22.857	1640	81.487	24.0
440	77.468	24.0	1060	76.814	22.857	1660	81.574	24.0
460	77.865	24.0	1080	76.882	22.857	1680	81.647	24.0
480	78.308	24.0	1100	76.962	22.857	1700	81.643	24.0
500	77.805	23.619	1120	79.26	22.857	1720	81.609	24.0
520	78.095	23.619	1140	80.266	24.0	1740	81.585	24.0
540	78.575	24.0	1160	80.351	24.0	1760	81.482	24.0
560	78.684	24.0	1180	80.361	24.0	1780	81.431	24.0
580	78.769	24.0	1200	80.397	24.0	1800	81.443	24.0
600	78.818	24.0						

The second experiment was again on the cave environment. The average of twenty runs can be seen in Table 5.3. Coverage increases from the start to about an average of $82.244m^2$ coverage from a starting coverage of $35.372m^2$. After 300 seconds, the average coverage was $74.775m^2$, and 600 seconds, $79.05m^2$. Thus, at least with this particular setup, roughly 84.07% of the expansion is done by 300 seconds, and by 600 seconds, roughly 93.23% of the final coverage is completed. Once again, evidence that the algorithm occasionally disconnected was manifested by less than perfect 24.0 values in the coverage column– at a couple of points the algorithm caused the robots to be disconnected, although this was transient behavior. There was little or no difference between the fixed and randomized cases, which causes us to conclude that at least in terms of aggregate behavior, the algorithm was not sensitive to minor positional variations.

Figure 5.8: Randomized Cave, Coverage over Time

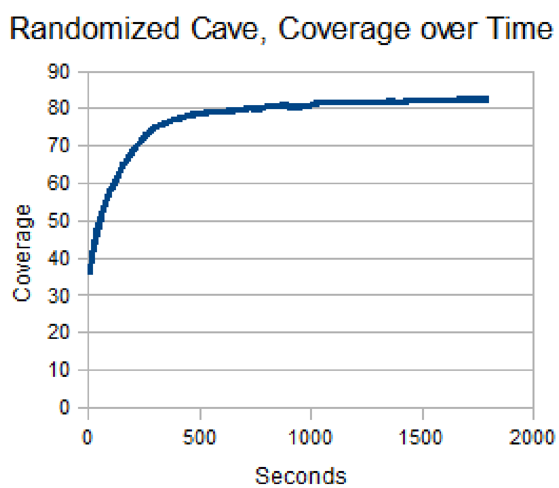


Table 5.3: Cave, Randomized Positions

Time	**Cov.** (m^2)	**Con.**	**Time**	**Cov.** (m^2)	**Con.**	**Time**	**Cov.** (m^2)	**Con.**
0	35.372	24.0	620	79.05	24.0	1220	81.46	24.0
20	41.485	24.0	640	79.108	24.0	1240	81.461	24.0
40	47.598	24.0	660	79.225	24.0	1260	81.476	24.0
60	52.118	24.0	680	79.378	24.0	1280	81.505	24.0
80	55.255	24.0	700	79.53	24.0	1300	81.539	24.0
100	57.939	24.0	720	79.65	24.0	1320	81.562	24.0
120	60.439	24.0	740	79.818	24.0	1340	81.609	24.0
140	62.874	24.0	760	79.971	24.0	1360	81.667	24.0
160	65.114	24.0	780	79.703	23.81	1380	81.743	24.0
180	67.07	24.0	800	79.79	23.81	1400	81.789	23.619
200	68.746	24.0	820	80.355	24.0	1420	81.303	23.619
220	70.247	24.0	840	80.436	24.0	1440	81.732	23.619
240	71.682	24.0	860	80.585	24.0	1460	82.0	24.0
260	72.854	24.0	880	80.696	24.0	1480	82.054	24.0
280	73.914	24.0	900	80.779	24.0	1500	82.065	24.0
300	74.775	24.0	920	80.622	23.81	1520	82.059	24.0
320	75.427	24.0	940	80.202	23.81	1540	82.065	24.0
340	75.945	24.0	960	80.244	23.81	1560	82.084	24.0
360	76.405	24.0	980	80.314	23.81	1580	82.1	24.0
380	76.839	24.0	1000	80.409	23.81	1600	82.11	24.0
400	77.194	24.0	1020	80.801	23.81	1620	82.105	24.0
420	77.526	24.0	1040	81.228	24.0	1640	82.084	24.0
440	77.869	24.0	1060	81.324	24.0	1660	82.095	24.0
460	78.164	24.0	1080	81.373	24.0	1680	82.128	24.0
480	78.289	24.0	1100	81.389	24.0	1700	82.154	24.0
500	78.468	24.0	1120	81.399	24.0	1720	82.179	24.0
520	78.631	24.0	1140	81.412	24.0	1740	82.206	24.0
540	78.764	24.0	1160	81.43	24.0	1760	82.197	24.0
560	78.895	24.0	1180	81.449	24.0	1780	82.229	24.0
580	79.003	24.0	1200	81.462	24.0	1800	82.244	24.0
600	79.05	24.0						

The third experiment was run on the Autolab environment. The final dispersion achieved a coverage of roughly $99.343m^2$. After 300 seconds, the dispersion was $80.791m^2$, which represented 66.96% of the total dispersion; after 600 seconds, the total dispersion was roughly $88.564m^2$ which was 80.98% of the total dispersion. None of the runs disconnected at any point.

Figure 5.9: Randomized Autolab, Coverage over Time

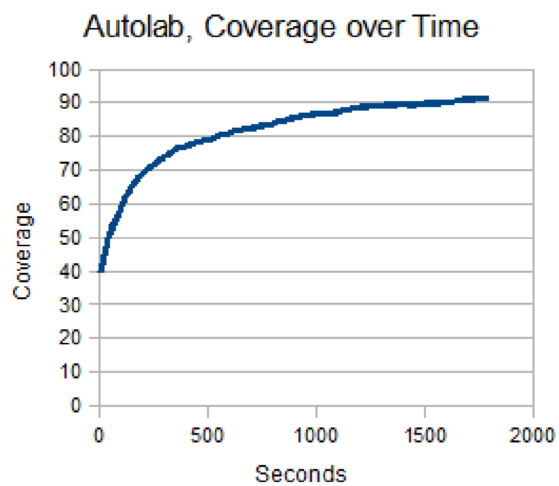


Table 5.4: Autolab, Randomized Positions

Time	**Cov. (m^2)**	**Con.**	**Time**	**Cov. (m^2)**	**Con.**	**Time**	**Cov. (m^2)**	**Con.**
0	43.189	24.0	620	88.564	24.0	1220	96.699	24.0
20	48.723	24.0	640	88.964	24.0	1240	96.801	24.0
40	54.257	24.0	660	89.314	24.0	1260	96.91	24.0
60	58.361	24.0	680	89.641	24.0	1280	97.018	24.0
80	61.502	24.0	700	89.863	24.0	1300	97.042	24.0
100	64.703	24.0	720	90.005	24.0	1320	97.144	24.0
120	67.951	24.0	740	90.315	24.0	1340	97.31	24.0
140	70.5	24.0	760	90.688	24.0	1360	97.387	24.0
160	72.515	24.0	780	90.962	24.0	1380	97.398	24.0
180	74.153	24.0	800	91.198	24.0	1400	97.453	24.0
200	75.529	24.0	820	91.573	24.0	1420	97.471	24.0
220	76.701	24.0	840	91.979	24.0	1440	97.45	24.0
240	77.819	24.0	860	92.288	24.0	1460	97.406	24.0
260	78.889	24.0	880	92.707	24.0	1480	97.578	24.0
280	79.845	24.0	900	93.115	24.0	1500	97.831	24.0
300	80.791	24.0	920	93.383	24.0	1520	98.01	24.0
320	81.799	24.0	940	93.678	24.0	1540	98.071	24.0
340	82.726	24.0	960	93.826	24.0	1560	98.006	24.0
360	83.543	24.0	980	94.0	24.0	1580	98.001	24.0
380	83.781	23.81	1000	94.274	24.0	1600	98.091	24.0
400	84.44	23.81	1020	94.397	24.0	1620	98.234	24.0
420	85.019	24.0	1040	94.46	24.0	1640	98.441	24.0
440	85.273	24.0	1060	94.599	24.0	1660	98.689	24.0
460	85.517	24.0	1080	94.683	24.0	1680	98.926	24.0
480	85.85	24.0	1100	94.919	24.0	1700	99.116	24.0
500	86.273	24.0	1120	95.249	24.0	1720	99.21	24.0
520	86.755	24.0	1140	95.61	24.0	1740	99.223	24.0
540	87.249	24.0	1160	95.964	24.0	1760	99.26	24.0
560	87.667	24.0	1180	96.226	24.0	1780	99.279	24.0
580	88.032	24.0	1200	96.532	24.0	1800	99.343	24.0
600	88.564	24.0						

The next experiment was the house environment. Here, coverage increased from $37.792m^2$ at the start to $73.995m^2$ after 300 seconds, $75.867m^2$ after 600 seconds, and $76.128m^2$ after 3600 seconds. Thus, by 300 seconds, 94.44% of the expansion was complete; by 600 seconds, 99.3% was finished. In this environment, the robots spread out almost entirely in the original room, and all but ignored the adjoining room. The final coverage of $76.128m^2$ was also short of the $160.27m^2$ obtained by the incremental greedy algorithm.

Figure 5.10: Randomized House, Coverage over Time

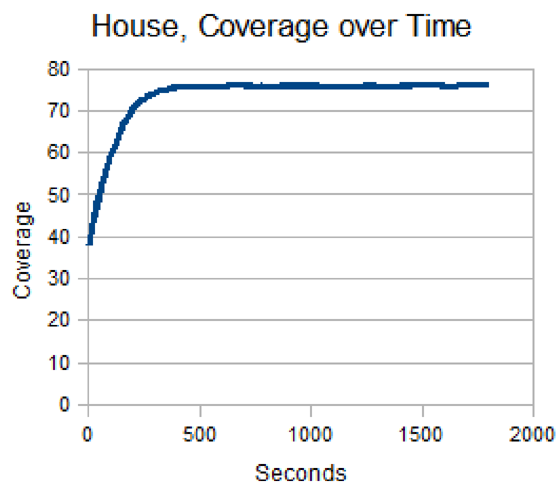
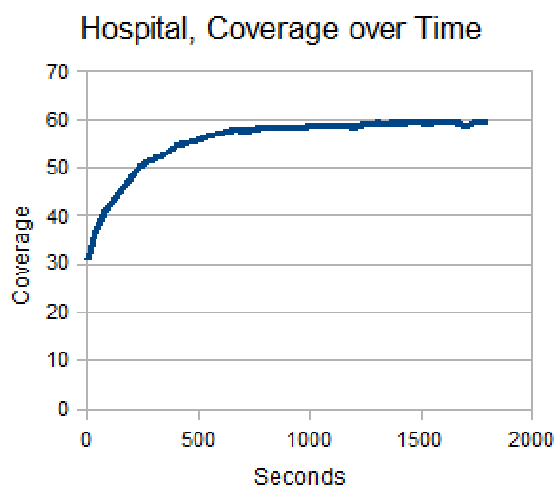


Table 5.5: House, Randomized Positions

Time	Cov. (m^2)	Con.	Time	Cov. (m^2)	Con.	Time	Cov. (m^2)	Con.
0	37.792	24.0	620	75.687	24.0	1220	75.516	24.0
20	43.001	24.0	640	75.798	24.0	1240	75.613	24.0
40	48.209	24.0	660	75.847	24.0	1260	75.708	24.0
60	52.543	24.0	680	75.896	24.0	1280	75.769	24.0
80	55.916	24.0	700	75.9	24.0	1300	75.825	24.0
100	58.819	24.0	720	75.83	24.0	1320	75.723	24.0
120	61.754	24.0	740	75.791	24.0	1340	75.651	24.0
140	64.406	24.0	760	75.743	24.0	1360	75.645	24.0
160	66.776	24.0	780	75.681	24.0	1380	75.658	24.0
180	68.713	24.0	800	75.704	24.0	1400	75.674	24.0
200	70.243	24.0	820	75.634	24.0	1420	75.703	24.0
220	71.481	24.0	840	75.572	24.0	1440	75.733	24.0
240	72.369	24.0	860	75.625	24.0	1460	75.758	24.0
260	73.107	24.0	880	75.743	24.0	1480	75.855	24.0
280	73.529	23.905	900	75.772	24.0	1500	75.948	24.0
300	73.995	23.905	920	75.799	24.0	1520	75.978	24.0
320	74.509	24.0	940	75.762	24.0	1540	75.993	24.0
340	74.818	24.0	960	75.732	24.0	1560	76.011	24.0
360	75.164	24.0	980	75.739	24.0	1580	76.051	24.0
380	75.344	24.0	1000	75.715	24.0	1600	75.919	23.857
400	75.465	24.0	1020	75.708	24.0	1620	75.747	23.857
420	75.508	24.0	1040	75.723	24.0	1640	75.62	23.714
440	75.529	24.0	1060	75.728	24.0	1660	75.646	23.714
460	75.518	24.0	1080	75.686	24.0	1680	76.003	23.857
480	75.49	24.0	1100	75.646	24.0	1700	76.023	24.0
500	75.425	24.0	1120	75.636	24.0	1720	76.017	24.0
520	75.474	24.0	1140	75.566	24.0	1740	76.019	24.0
540	75.554	24.0	1160	75.483	24.0	1760	76.06	24.0
560	75.643	24.0	1180	75.439	24.0	1780	76.098	24.0
580	75.633	24.0	1200	75.449	24.0	1800	76.128	24.0
600	75.687	24.0						

The final tested environment was the hospital environment. This one also highlighted the inadequacies of this particular algorithm in adequately handling narrow doorways. The robots in the hallway did manage to spread out, but robots in the room for the most part never managed to escape the room. Frequently, the doorway would become blocked during the course of a run, and that would prevent robots from leaving. The final expansion was on average around $59.205m^2$. after 30 minutes, starting from an initial coverage of 30.833. After 300 seconds, the algorithm covered $51.95m^2$, representing 74.43% of the total expansion. After 10 minutes, the average coverage was $57.008m^2$, which represented an average of 92.26% of the total expansion achieved by the algorithm. The total expansion that the algorithm achieved, $59.205m^2$ was also well short of the maximum $146.759m^2$ that the Incremental Greedy Algorithm obtained in this environment, in large part because of an inability to get many robots out of the starting room.

Figure 5.11: Randomized Hospital, Coverage over Time



The algorithm had some similar characteristics across all environments. Most of the coverage obtained was obtained in the first 5 to 10 minutes, with further coverage gains after that point being very small. In no case did the algorithm outperform the incremental greedy algorithm or approach the theoretical maximum; results that were not completely surprising, due to the fact that the incremental greedy algorithm is a sequential and exhaustive approach to the coverage problem. Our approach also did not do well in complex environments with doorways and enclosed rooms– the hospital environment in particular was quite difficult for this approach. As mentioned before, robots already in the hallway fared well, while robots unfortunate enough to be in the

Table 5.6: Hospital, Randomized Positions

Time	Cov. (m^2)	Con.	Time	Cov. (m^2)	Con.	Time	Cov. (m^2)	Con.
0	30.833	24.0	620	57.008	24.0	1220	58.299	23.905
20	33.73	24.0	640	57.306	24.0	1240	58.336	23.905
40	36.628	24.0	660	57.506	24.0	1260	58.816	23.905
60	39.008	24.0	680	57.6	24.0	1280	58.956	24.0
80	40.767	24.0	700	57.578	24.0	1300	58.978	24.0
100	42.092	24.0	720	57.512	24.0	1320	59.022	24.0
120	43.175	24.0	740	57.542	24.0	1340	59.049	24.0
140	44.375	24.0	760	57.72	24.0	1360	59.008	24.0
160	45.676	24.0	780	57.87	24.0	1380	59.045	24.0
180	46.978	24.0	800	57.955	24.0	1400	59.113	24.0
200	48.118	24.0	820	57.998	24.0	1420	59.109	24.0
220	49.341	24.0	840	58.072	24.0	1440	59.075	24.0
240	50.283	24.0	860	58.049	24.0	1460	59.186	24.0
260	51.024	24.0	880	58.048	24.0	1480	59.224	24.0
280	51.564	24.0	900	58.122	24.0	1500	59.206	24.0
300	51.95	24.0	920	58.192	24.0	1520	59.153	24.0
320	52.188	24.0	940	58.101	24.0	1540	59.084	24.0
340	52.659	24.0	960	58.127	24.0	1560	59.076	24.0
360	53.158	24.0	980	58.219	24.0	1580	59.156	24.0
380	53.664	24.0	1000	58.292	24.0	1600	59.238	24.0
400	54.369	24.0	1020	58.363	24.0	1620	59.252	24.0
420	54.75	24.0	1040	58.509	24.0	1640	59.275	24.0
440	54.97	24.0	1060	58.504	24.0	1660	59.276	24.0
460	55.23	24.0	1080	58.438	24.0	1680	59.258	24.0
480	55.48	24.0	1100	58.459	24.0	1700	59.134	23.762
500	55.789	24.0	1120	58.511	24.0	1720	58.475	23.762
520	56.116	24.0	1140	58.529	24.0	1740	59.017	23.762
540	56.386	24.0	1160	58.565	24.0	1760	59.354	24.0
560	56.573	24.0	1180	58.683	24.0	1780	59.237	24.0
580	56.772	24.0	1200	58.578	23.905	1800	59.205	24.0
600	57.008	24.0						

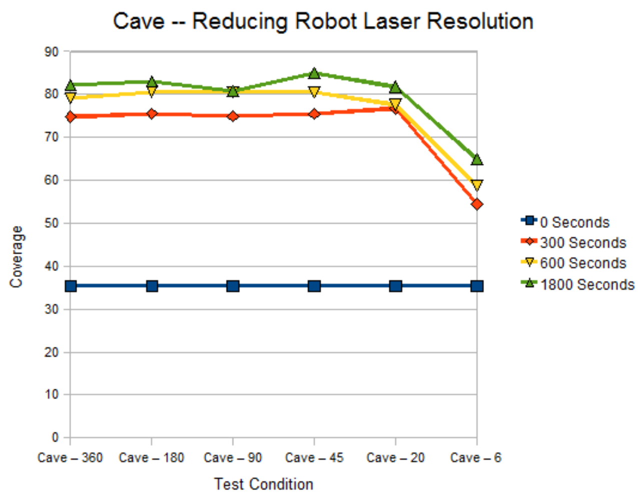
room were unsuccessful in leaving the room, which reduced the efficacy of the overall approach to the coverage problem on that environment.

5.2.8 Dispersion with Fewer Laser Measurements

The next set of experiments that we performed was with reducing the number of laser readings available to each robot. We tried several different conditions, the previous

360-measurement results in the last section, as well as experiments with 180, 90, 45, 20, and 6 measurements. In each of these cases, the measurements are equidistant. The algorithm appears to be fairly robust to a reduction in resolution; only when the number of laser readings was reduced to 6 was there a significant decline in observed coverage. However, even in that case, the the robots were able too disperse to cover 183.42% of their initial coverage indicating that even very limited robots may succeed at dispersing somewhat, although not as much as with higher readings.

Figure 5.12: Reducing Laser Readings



A summary of the results of reducing the laser resolution for robots in the cave environment is above.

Table 5.7: Cave, Randomized, 180 Readings

Time	**Cov. (m^2)**	**Con.**	**Time**	**Cov. (m^2)**	**Con.**	**Time**	**Cov. (m^2)**	**Con.**
0	35.372	24.0	620	80.584	24.0	1220	81.297	23.714
20	41.244	24.0	640	80.738	24.0	1240	81.316	23.714
40	47.115	24.0	660	80.84	24.0	1260	81.285	23.714
60	51.627	24.0	680	80.902	24.0	1280	81.278	23.714
80	54.895	24.0	700	80.96	24.0	1300	81.336	23.714
100	57.687	24.0	720	81.018	24.0	1320	81.344	23.714
120	60.322	24.0	740	81.076	24.0	1340	81.445	23.714
140	62.975	24.0	760	81.172	24.0	1360	81.512	23.714
160	65.282	24.0	780	81.318	24.0	1380	81.517	23.714
180	67.077	23.714	800	81.362	24.0	1400	81.492	23.714
200	68.416	23.714	820	81.457	24.0	1420	81.564	23.714
220	70.832	23.714	840	81.58	24.0	1440	80.645	23.429
240	72.292	24.0	860	81.603	24.0	1460	80.322	23.429
260	73.497	24.0	880	81.662	24.0	1480	80.841	23.429
280	74.561	24.0	900	81.685	24.0	1500	81.741	23.714
300	75.552	24.0	920	81.71	24.0	1520	81.763	23.714
320	76.429	24.0	940	81.773	24.0	1540	81.8	23.714
340	77.184	24.0	960	81.853	24.0	1560	81.89	23.714
360	77.733	24.0	980	81.906	24.0	1580	81.938	23.714
380	78.044	24.0	1000	81.94	24.0	1600	81.981	23.714
400	78.3	24.0	1020	82.018	24.0	1620	82.037	23.714
420	78.615	24.0	1040	82.079	24.0	1640	82.108	23.714
440	79.019	24.0	1060	82.121	24.0	1660	82.181	23.714
460	79.398	24.0	1080	82.159	24.0	1680	82.351	23.714
480	79.694	24.0	1100	82.191	24.0	1700	82.583	23.714
500	79.591	23.905	1120	82.163	24.0	1720	82.485	23.571
520	79.839	23.905	1140	81.711	23.714	1740	82.488	23.571
540	80.161	24.0	1160	81.174	23.714	1760	82.922	23.714
560	80.308	24.0	1180	81.113	23.714	1780	82.999	23.714
580	80.435	24.0	1200	81.19	23.714	1800	83.056	23.714
600	80.584	24.0						

Table 5.8: Cave, 90 Readings

Time	Cov. (m^2)	Con.	Time	Cov. (m^2)	Con.	Time	Cov. (m^2)	Con.
0	35.374	24.0	620	80.572	23.857	1220	82.357	24.0
20	40.404	24.0	640	80.392	23.857	1240	82.421	24.0
40	45.435	24.0	660	80.458	23.857	1260	82.484	24.0
60	49.69	24.0	680	80.942	23.857	1280	82.523	24.0
80	53.091	24.0	700	81.225	24.0	1300	82.523	24.0
100	56.108	24.0	720	81.326	24.0	1320	82.603	24.0
120	58.928	24.0	740	81.149	23.81	1340	82.679	24.0
140	61.706	24.0	760	80.751	23.81	1360	82.749	24.0
160	64.314	24.0	780	81.275	23.81	1380	82.756	24.0
180	66.739	24.0	800	81.372	24.0	1400	82.685	24.0
200	68.946	24.0	820	81.433	24.0	1420	82.647	24.0
220	70.736	24.0	840	81.453	24.0	1440	82.673	24.0
240	72.074	24.0	860	81.43	24.0	1460	82.668	24.0
260	73.174	24.0	880	81.458	24.0	1480	82.653	24.0
280	74.102	24.0	900	81.543	24.0	1500	82.222	23.286
300	74.929	24.0	920	81.593	24.0	1520	80.681	23.286
320	75.729	24.0	940	81.599	24.0	1540	80.72	23.286
340	76.51	24.0	960	81.414	23.762	1560	80.777	23.286
360	77.309	24.0	980	80.903	23.762	1580	80.816	23.286
380	77.971	24.0	1000	81.636	23.762	1600	80.799	23.286
400	78.458	24.0	1020	81.755	24.0	1620	80.723	23.286
420	78.799	24.0	1040	81.795	24.0	1640	80.636	23.286
440	79.089	24.0	1060	81.879	24.0	1660	80.586	23.286
460	79.297	24.0	1080	81.96	24.0	1680	80.621	23.286
480	79.233	23.762	1100	81.937	24.0	1700	80.536	23.143
500	79.397	23.762	1120	81.864	24.0	1720	80.339	23.143
520	80.038	24.0	1140	81.864	24.0	1740	80.609	23.143
540	80.283	24.0	1160	81.929	24.0	1760	80.668	23.286
560	80.477	24.0	1180	82.058	24.0	1780	80.754	23.286
580	80.632	24.0	1200	82.286	24.0	1800	80.8	23.286
600	80.572	23.857						

Table 5.9: Cave, 45 Readings

Time	**Cov. (m^2)**	**Con.**	**Time**	**Cov. (m^2)**	**Con.**	**Time**	**Cov. (m^2)**	**Con.**
0	35.371	24.0	620	80.539	23.667	1220	84.115	24.0
20	41.066	24.0	640	80.711	23.476	1240	84.235	24.0
40	46.762	24.0	660	80.709	23.81	1260	84.368	24.0
60	51.191	24.0	680	80.974	23.81	1280	84.489	24.0
80	54.488	24.0	700	81.606	24.0	1300	84.549	24.0
100	57.317	24.0	720	81.682	24.0	1320	84.597	24.0
120	60.002	24.0	740	81.686	24.0	1340	84.635	24.0
140	62.556	24.0	760	81.749	24.0	1360	84.665	24.0
160	64.75	24.0	780	81.844	24.0	1380	84.707	24.0
180	66.751	24.0	800	81.92	24.0	1400	84.746	24.0
200	68.733	24.0	820	81.899	24.0	1420	84.771	24.0
220	70.46	24.0	840	81.844	24.0	1440	84.76	24.0
240	72.077	24.0	860	81.808	24.0	1460	84.71	24.0
260	73.42	24.0	880	81.85	24.0	1480	84.672	24.0
280	74.54	24.0	900	81.963	24.0	1500	84.671	24.0
300	75.469	24.0	920	82.1	24.0	1520	84.731	24.0
320	76.247	24.0	940	82.256	24.0	1540	84.759	24.0
340	76.763	24.0	960	82.379	24.0	1560	84.831	24.0
360	77.048	24.0	980	82.495	24.0	1580	84.811	24.0
380	77.496	24.0	1000	82.177	23.762	1600	84.793	24.0
400	77.944	24.0	1020	82.017	23.762	1620	84.817	24.0
420	78.377	24.0	1040	82.076	23.762	1640	84.87	24.0
440	78.749	23.857	1060	82.23	23.762	1660	84.907	24.0
460	78.954	23.857	1080	82.984	23.762	1680	84.896	24.0
480	79.475	23.857	1100	83.222	24.0	1700	84.905	24.0
500	80.232	23.857	1120	83.38	24.0	1720	84.88	24.0
520	80.682	24.0	1140	83.534	24.0	1740	84.867	24.0
540	80.75	24.0	1160	83.7	24.0	1760	84.944	24.0
560	80.851	24.0	1180	83.873	24.0	1780	84.973	24.0
580	81.019	23.667	1200	84.008	24.0	1800	85.019	24.0
600	80.539	23.667						

Table 5.10: Cave, 20 Readings

Time	Cov. (m^2)	Con.	Time	Cov. (m^2)	Con.	Time	Cov. (m^2)	Con.
0	35.372	24.0	620	77.586	22.524	1220	77.798	22.762
20	40.712	24.0	640	77.652	22.095	1240	77.77	22.762
40	46.053	24.0	660	76.44	22.095	1260	76.897	22.143
60	50.271	24.0	680	76.54	22.095	1280	76.578	22.143
80	53.387	24.0	700	76.651	22.095	1300	77.615	22.762
100	56.115	24.0	720	76.705	22.095	1320	77.62	22.762
120	58.923	24.0	740	76.297	21.905	1340	77.583	22.762
140	61.579	24.0	760	76.465	21.905	1360	77.529	22.762
160	64.105	24.0	780	76.662	22.095	1380	77.545	22.762
180	66.467	24.0	800	76.71	22.095	1400	77.551	22.762
200	68.651	23.905	820	76.812	22.095	1420	77.566	22.762
220	70.441	23.905	840	76.974	22.095	1440	77.577	22.762
240	72.443	23.905	860	77.123	22.095	1460	77.535	22.762
260	74.143	24.0	880	77.245	22.095	1480	77.196	22.524
280	75.553	24.0	900	77.034	21.81	1500	76.927	22.524
300	76.766	24.0	920	80.074	21.81	1520	78.934	22.762
320	77.823	24.0	940	80.72	23.286	1540	80.093	22.762
340	78.744	24.0	960	80.735	23.286	1560	80.328	22.857
360	79.393	24.0	980	80.839	23.333	1580	80.324	22.857
380	79.963	24.0	1000	80.847	23.333	1600	80.323	22.857
400	80.497	24.0	1020	80.883	23.333	1620	80.319	22.857
420	80.989	24.0	1040	81.505	23.333	1640	80.335	22.857
440	81.309	24.0	1060	81.622	23.619	1660	80.317	22.857
460	81.445	24.0	1080	79.74	22.381	1680	80.68	22.857
480	81.554	24.0	1100	78.221	22.381	1700	82.065	23.81
500	81.407	23.762	1120	78.823	22.571	1720	81.871	22.952
520	79.231	22.429	1140	78.784	22.667	1740	79.93	22.952
540	78.005	22.429	1160	79.282	22.667	1760	81.127	22.952
560	81.336	22.429	1180	78.546	22.762	1780	81.93	23.571
580	77.969	22.524	1200	77.831	22.762	1800	81.789	23.143
600	77.586	22.524						

Table 5.11: Cave, 6 Readings

Time	**Cov.** (m^2)	**Con.**	**Time**	**Cov.** (m^2)	**Con.**	**Time**	**Cov.** (m^2)	**Con.**
0	35.374	24.0	620	58.722	24.0	1220	63.223	24.0
20	38.309	24.0	640	58.986	24.0	1240	63.334	24.0
40	41.245	24.0	660	59.279	24.0	1260	63.427	24.0
60	43.629	24.0	680	59.582	24.0	1280	63.505	24.0
80	45.356	24.0	700	59.855	24.0	1300	63.569	24.0
100	46.673	24.0	720	59.999	24.0	1320	63.635	24.0
120	47.81	24.0	740	60.107	24.0	1340	63.686	24.0
140	48.596	24.0	760	60.173	24.0	1360	63.724	24.0
160	49.284	24.0	780	60.254	24.0	1380	63.762	24.0
180	50.017	24.0	800	60.314	24.0	1400	63.832	24.0
200	50.82	24.0	820	60.36	24.0	1420	63.926	24.0
220	51.665	24.0	840	60.43	24.0	1440	64.011	24.0
240	52.462	24.0	860	60.51	24.0	1460	64.104	24.0
260	53.142	24.0	880	60.621	24.0	1480	64.197	24.0
280	53.793	24.0	900	60.756	24.0	1500	64.286	24.0
300	54.388	24.0	920	60.908	24.0	1520	64.381	24.0
320	54.934	24.0	940	61.008	24.0	1540	64.474	24.0
340	55.359	24.0	960	61.17	24.0	1560	64.563	24.0
360	55.731	24.0	980	61.327	24.0	1580	64.611	24.0
380	56.025	24.0	1000	61.534	24.0	1600	64.624	24.0
400	56.306	24.0	1020	61.741	24.0	1620	64.627	24.0
420	56.604	24.0	1040	61.941	24.0	1640	64.622	24.0
440	56.914	24.0	1060	62.171	24.0	1660	64.612	24.0
460	57.23	24.0	1080	62.389	24.0	1680	64.612	24.0
480	57.55	24.0	1100	62.552	24.0	1700	64.612	24.0
500	57.831	24.0	1120	62.684	24.0	1720	64.621	24.0
520	58.034	24.0	1140	62.805	24.0	1740	64.653	24.0
540	58.196	24.0	1160	62.93	24.0	1760	64.726	24.0
560	58.322	24.0	1180	63.028	24.0	1780	64.844	24.0
580	58.509	24.0	1200	63.12	24.0	1800	64.885	24.0
600	58.722	24.0						

Chapter 6

Conclusions & Future Work

We have created a modular, reusable Python framework in which to launch experiments utilizing the Player/Stage environment. We have proposed a new algorithm, the Alarm Counting Hops algorithm, for distributed robotic dispersion using an emergent semi-centralized method that utilizes leader-election, counting-hops, and potential fields techniques to disperse nodes and construct a mobile sensor network. We have demonstrated within the virtual Player-Stage framework that this approach can achieve over a 100% increase in coverage, while maintaining connectivity, given a densely packed robot swarm with only wireless communication and laser-rangefinding capabilities. We have also shown that this approach is robust to small changes in initial conditions, that the approach can be applied effectively on robots with laser sensors with significantly reduced resolution, and that it is applicable to different types of environments, although not surprisingly, the Alarm Counting Hops algorithm fares better in more open environments.

As always, there are new and exciting directions for future research, some of which we will detail below:

6.1 Parametrization

As mentioned in the discussion, we found parametrization to be extremely important in the context of this algorithmic approach. We do not mean to suggest in any way that the parameters we chose in our experimental design were by any means optimal or even near-optimal to maximize any characteristic. As such, a more measured search approach, such as that provided by a learning algorithm or an evolutionary approach would likely greatly improve the dispersion algorithm and performance across environments.

6.2 Sensing Behaviors

Another extension of our work would be to incorporate visual/sonar data from the robotic sensors and use that information to allow robots to move more intelligently. In this work, a robot will blindly walk into a wall if the forces acting on it are appropriately large. They also do not distinguish robots in their visual field from other obstacles. This would help deal with some of the issues presented by the narrow openings mentioned above and help robots to spread out in a more coordinated fashion. However, this does come at a cost: the robots would be more expensive if they were to have more advanced capabilities.

6.3 Directed Dispersion

A further extension would be perhaps attempting to incorporate some of the ideas used in [31] regarding directed dispersion in swarms. It may be possible to achieve good results by having robots disperse blindly at first, and then have them repair the structure and fix holes by using directed dispersion. By adding appropriate rules, this may also help to reduce the chaotic effects that result from robots attempting to escape a well populated area and help channel them into more productive areas. More generally, communication of some kind can be used to improve the dispersion of the network, as in our approach, robots can neither discern the locations of other robots, let alone use such cues to direct dispersion.

6.4 Real World Robots

A final domain that we would like to pursue in the future is actually implementing this approach with real robots. It is not unreasonable to expect that there will be significant hurdles to successfully doing so; basic assumptions such as no noise and perfect communication are unlikely to hold. Additionally, signal strength is likely to be somewhat patchy, and not degrade uniformly throughout the environment, particularly when obstacles are involved. Robots in our algorithm depend on signal strength to determine whether they can move or not, so it would likely have some implications for connectivity.

Bibliography

[1] Noa Agmon, Noam Hazon, and Gal A. Kaminka, *Constructing spanning trees for efficient multi-robot coverage*, IEEE International Conference on Robotics and Automation 2006, 2006.

[2] ———, *The giving tree: Constructing trees for efficient offline and online multi-robot coverage*, Annals of Mathematics and Artificial Intelligence **52** (2008), no. 2–4, 143–168.

[3] Hüseyin Akcan, Vassil Kriakov, Hervé Brönnimann, and Alex Delis, *Gps-free node localization in mobile wireless sensor networks*, MobiDE '06: Proceedings of the 5th ACM international workshop on Data engineering for wireless and mobile access (New York, NY, USA), ACM, 2006, pp. 35–42.

[4] Ronald C. Arkin, *Motor schema based navigation for a mobile robot: An approach to programming by behavior*, 1987 IEEE Conference on Robotics and Automation, vol. 4, 1987, pp. 264–271.

[5] Jonathan Bachrach, Radhika Nagpal, Michael Salib, and Howard E. Shrobe, *Experimental results for and theoretical analysis of a self-organizing global coordinate system for ad hoc sensor networks*, Telecommunication Systems **26** (2004), no. 2-4, 213–233.

[6] Laura Barnes, Wendy Alvis, MaryAnne Fields, Kimon Valavanis, and Wilfrido Moreno, *Heterogeneous swarm formation control using bivariate normal functions to generate potential fields*, DIS '06: Proceedings of the IEEE Workshop on Distributed Intelligent Systems: Collective Intelligence and Its Applications (DIS'06) (Washington, DC, USA), IEEE Computer Society, 2006, pp. 85–94.

[7] Maxim Batalin and Gaurav S. Sukhatme, *Spreading out: A local approach to multi-robot coverage*, Proceedings of the International Symposium on Distributed Autonomous Robotic Systems (Fukuoka, Japan), Jun 2002, pp. 373–382.

[8] ———, *Dynamic coverage via multi-robot cooperation*, Proceedings from the International Workshop on Multi-Robot Systems (Washington DC) (A. Schultz, L. E. Parker, and F. Schneider, eds.), Kluwer Academic Publishers, Mar 2003, pp. 295–296.

[9] ———, *The design and analysis of an efficient local algorithm for coverage and exploration based on sensor network deployment*, IEEE Transactions on Robotics **23** (2007), no. 4, 661–675.

[10] Rodney A. Brooks, *A robust layered control system for a mobile robot*, Tech. report, Cambridge, MA, USA, 1985.

[11] M. Broxvall, S. Coradeschi, A. Loutfi, and A. Saffiotti, *An ecological approach to odour recognition in intelligent environments*, Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on (2006), 2066–2071.

[12] Howie Choset, *Coverage for robotics – a survey of recent results*, Annals of Mathematics and Artificial Intelligence **31** (2001), no. 1-4, 113–126.

[13] Vin de Silva, Robert Ghrist, and Abubakr Muhammad, *Blind swarms for coverage in 2-d*, Robotics: Science and Systems, 2005, pp. 335–342.

[14] Yoav Gabriely and Elon Rimon, *Spanning-tree based coverage of continuous areas by a mobile robot*, Annals of Mathematics and Artificial Intelligence **31** (2001), no. 1-4, 77–98.

[15] Douglas W. Gage, *Command control for many-robot systems*, AUVS-92, 1992.

[16] B. Gerkey, R. Vaughan, and A. Howard, *The player/stage project: Tools for multi-robot and distributed sensor systems*, 11th International Conference on Advanced Robotics, 2003.

[17] B.P. Gerkey, R.T. Vaughan, K. Stoy, A. Howard, G.S. Sukhatme, and M.J. Mataric, *Most valuable player: a robot device server for distributed control*, Intelligent Robots and Systems, 2001. Proceedings. 2001 IEEE/RSJ International Conference on **3** (2001), 1226–1231 vol.3.

[18] Andreas Haeberlen, Eliot Flannery, Andrew M. Ladd, Algis Rudys, Dan S. Wallach, and Lydia E. Kavraki, *Practical robust localization over large-scale 802.11 wireless networks*, MobiCom '04: Proceedings of the 10th annual international

conference on Mobile computing and networking (New York, NY, USA), ACM, 2004, pp. 70–84.

[19] Noam Hazon and Gal A. Kaminka, *Redundancy, efficiency, and robustness in multi-robot coverage*, IEEE International Conference on Robotics and Automation 2005, 2005.

[20] Noam Hazon, Fabrizio Mieli, and Gal A. Kaminka, *Towards robust on-line multi-robot coverage*, IEEE International Conference on Robotics and Automation 2006, 2006.

[21] Andrew Howard, Maja J. Matarić, and Gaurav S. Sukhatme, *An incremental self-deployment algorithm for mobile sensor networks*, Autonomous Robots Special Issue on Intelligent Embedded Systems **13** (2002), no. 2, 113–126.

[22] ———, *Mobile sensor network deployment using potential fields: A distributed, scalable solution to the area coverage problem*, Proceedings of the International Symposium on Distributed Autonomous Robotic Systems, 2002, pp. 299–308.

[23] Andrew Howard, Lynne E. Parker, and Gaurav S. Sukhatme, *Experiments with a large heterogeneous mobile robot team: Exploration, mapping, deployment and detection*, Int. J. Rob. Res. **25** (2006), no. 5-6, 431–447.

[24] Andrew Howard, Sajid Siddiqi, and Gaurav S. Sukhatme, *An experimental study of localization using wireless ethernet*, Proceedings of the International Conference on Field and Service Robotics, Jul 2003.

[25] T.-R. Hsiang, E. M. Arkin, M. A. Bender, S. P. Fekete, and Joseph S. B. Mitchell, *Algorithms for rapidly dispersing robot swarms in unknown environments*, 5th International Workshop on Algorithmic Foundations of Robotics, 2001.

[26] T. Ishigaki, I. Okawa, K. Kobayashi, and K. Watanabe, *Alignment and uniform dispersion of multiple mobile robots in a room*, SICE 2003 Annual Conference **3** (2003), 2908–2911.

[27] Oussama Khatib, *Real-time obstacle avoidance for manipulators and mobile robots*, Int. J. Rob. Res. **5** (1986), no. 1, 90–98.

[28] Koen Langendoen and Niels Reijers, *Distributed localization in wireless sensor networks: a quantitative comparison*, Comput. Netw. **43** (2003), no. 4, 499–518.

[29] Amy Loutfi and Silvia Coradeschi, *Smell, think and act: A cognitive robot discriminating odours*, Auton. Robots **20** (2006), no. 3, 239–249.

[30] Luke Ludwig and Maria Gini, *Robotic swarm dispersion using wireless intensity signals*, Distributed Autonomous Robotic Systems 2007, 2006, pp. 135–144.

[31] James McLurkin, *Distributed algorithms for multi-robot systems*, IPSN '07: Proceedings of the 6th international conference on Information processing in sensor networks (New York, NY, USA), ACM, 2007, pp. 545–546.

[32] James McLurkin and Jennifer Smith, *Distributed algorithms for dispersion in indoor environments using a swarm of autonomous mobile robots*, Proceedings of the 2004 Distributed Autonomous Robotics Conference, 2004.

[33] Ryan Morlok and Maria Gini, *Dispersing robots in an unknown environment*, Distributed Autonomous Robotics Systems 2004, 2004.

[34] Eliyahu Osherovich, Vladimir Yanovki, Israel A. Wagner, and Alfred M. Bruckstein, *Robust and Efficient Covering of Unknown Continuous Domains with Simple, Ant-Like A(ge)nts*, The International Journal of Robotics Research **27** (2008), no. 7, 815–831.

[35] Lynne Parker, B. Kannan, Xiaoquan Fu, and Yifan Tang, *Heterogeneous mobile sensor net deployment using robot herding and line-of-sight formations*, Intelligent Robots and Systems, 2003. (IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on **3** (27-31 Oct. 2003), 2488–2493 vol.3.

[36] David Payton, Mike Daily, Regina Estowski, Mike Howard, and Craig Lee, *Pheromone robotics*, Auton. Robots **11** (2001), no. 3, 319–324.

[37] Janice L. Pearce, Paul E. Rybski, Sascha Stoeter, and Nikolaos Papanikolopoulos, *Dispersion behaviors for a team of multiple miniature robots*, IEEE International Conference on Robotics and Automation 2003, 2003, pp. 1158–1163.

[38] Sameera Poduri, Sundeep Pattem, Bhaskar Krishnamachari, and Gaurav S. Sukhatme, *Using local geometry for tunable topology control in sensor networks*, IEEE Transactions on Mobile Computing **8** (2009), no. 2, 218–230.

[39] Sameera Poduri and Gaurav S. Sukhatme, *Constrained coverage for mobile sensor networks*, IEEE International Conference on Robotics and Automation (New Orleans, LA), May 2004, pp. 165–172.

[40] K. Sugawara, T. Kazama, and T. Watanabe, *Foraging behavior of interacting robots with virtual pheromone*, Intelligent Robots and Systems, 2004. (IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on **3** (2004), 3074–3079 vol.3.

[41] Sebastian Thrun, *Robotic mapping: a survey*, Exploring artificial intelligence in the new millennium (G. Lakemeyer and B. Nebel, eds.), Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003, pp. 1–35.

[42] Emre Ugur, Ali E. Turgut, and Erol Sahin, *Dispersion of a swarm of robots based on realistic wireless intensity signals*, Proceedings of the 2007 Intenational Symposium on Computer and Information Science, 2007.

[43] Liang Yuan, Weidong Chen, and Yugeng Xi, *A review of control and localization for mobile sensor networks*, Intelligent Control and Automation, 2006. WCICA 2006. The Sixth World Congress on **2** (2006), 9164–9168.

Appendix

.1 Player Configuration File

```
# Desc: Player configuration file for controlling Stage devices
# load the Stage plugin simulation driver

driver
(
 name "stage"
 provides ["simulation:0"]
 plugin "libstageplugin"
 worldfile "pfvis.world"
)

driver
(
 name "stage"
 provides ["map:0"]
 model "cave"
)

driver
(
 name "stage"
 provides ["7000:position2d:0" "7000:laser:0" "7000:blobfinder:0" ]
 model "robot1"
)
```

```
...

driver
(
 name "stage"
 provides ["7011:position2d:0" "7011:laser:0" "7011:blobfinder:0" ]
 model "robot12"
)
```

.2 Player World File

```
# defines Pioneer-like robots
include "pioneer.inc"
# defines 'map' object used for floorplans
include "map.inc"
# defines sick laser
include "sick.inc"
# size of the world in meters
size [ 16 16 ]
# set the resolution of the underlying raytrace model in meters
resolution .02
# update the screen every 20ms
gui_interval 20

window
(
 size [591.000 638.000]
 center [-0.206 0.072]
 scale 0.028
)

# load an environment bitmap
map
(
```

```
 bitmap "cave.png"
 size [16 16]
 name "cave"
)


pioneer2dx
(
 name "robot1"
 color "red"
 pose [-3 0.6 -305.9]
 laser(
  samples 361
  range_min 0.0
  range_max 6.0
  fov 40.0
  color "blue"
  )
 ptz
 (
  blobfinder(
   channel_count 6
   channels ["red" "blue" "green" "cyan" "yellow" "magenta"]
   range_max 8.0
   size [0.01 0.01]
  )
 )
)

...

pioneer2dx
(
 name "robot12"
 color "red"
 pose [-3 0.6 -305.9]
```

```
 laser(
  samples 361
  range_min 0.0
  range_max 6.0
  fov 40.0
  color "blue"
  )
 ptz
 (
  blobfinder(
   channel_count 6
   channels ["red" "blue" "green" "cyan" "yellow" "magenta"]
   range_max 8.0
   size [0.01 0.01]
  )
 )
)
```

.3 Experiment File

```
<experiment experiment_runs="1" experiment_length="500"
 record_interval="100" sizex="16"  sizey="16"
 experiment_directory="experiments/pfvis" >
 <robot class="PotFieldRobotVis"
  xmlfile="experiments/pfvis/robotxml/robot1.xml"/>

...

 <robot class="PotFieldRobotVis"
  xmlfile="experiments/pfvis/robotxml/robot12.xml"/>
</experiment>
```

.4 Robot File

```
<robot>
<robotattribute name="ch_s" value="[’’]" type="list"/>
<robotattribute name="ch_hosts" value="[’’]" type="list"/>
<robotattribute name="ch_ports" value="[-1]" type="list"/>
 <robotattribute name="id" value="1" type="int"/>
 <robotattribute name="wireless_range" value="4" type="float"/>
 <robotattribute name="deviceport" value="7000" type="int"/>
 <robotattribute name="logfile" value="robot1.log" type="string"/>
 <robotattribute name="httpport" value="8000" type="int"/>
 <robotattribute name="robothost" value = "localhost" type="string"/>
 <robotattribute name="devicehost" value = "localhost" type="string"/>
 <robotattribute name="messagehost" value = "localhost" type="string"/>
 <robotattribute name="type" value = "1" type="int"/>
</robot>
```

.5 Experiment Setup File

```
<experiment name="pfvis" startdeviceport="7000" starthttpport="8000"
 worlddir="../worlds" sizex = "16" sizey = "16" resolution = ".02"
 experimentdirectory="experiments/pfvis" imagename="cave"
 imagefilename="cave.png" experiment_runs="1" experiment_length="500"
 record_interval = "100">
 <robottype number="12" color="red" pose="-3 0.6 -305.9"
  class="PotFieldRobotVis" wireless_range="4">
  <laser range_min="0.0" range_max="6.0" fov="40.0"
   samples="361" color="’blue’" size="0.14 0.14"/>
  <ptz>
   <blobfinder channel_count="6" channels="’red’ ’blue’ ’green’ ’cyan’
   ’yellow’ ’magenta’" range_max="8.0" size="0.01 0.01"/>
  </ptz>
  <robotxml>
   <robotattribute name="robothost" value="localhost" type="string"/>
   <robotattribute name="devicehost" value="localhost" type="string"/>
   <robotattribute name="messagehost" value="localhost" type="string"/>
```

```
    <robotattribute name="type" value="1" type="int"/>
    <robotattribute name="numberarobots" value = "4" type="int"/>
    <robotattribute name="numberbrobots" value = "8" type="int"/>
  </robotxml>
 </robottype>
</experiment>
```