An Incremental Self-Deployment Algorithm for Mobile Sensor Networks

Andrew Howard, Maja J Matarić and Gaurav S Sukhatme
Robotics Research Lab, Computer Science Department,
University of Southern California
E-mail: {ahoward@usc.edu, mataric@usc.edu, gaurav@usc.edu}

Abstract

This paper describes an incremental deployment algorithm for mobile sensor networks. A mobile sensor network is a distributed collection of nodes, each of which has sensing, computation, communication and locomotion capabilities. The algorithm deploys nodes one-at-a-time into an unknown environment, with each node making use of information gathered by previously deployed nodes to determine its target location. The algorithm is designed to maximize network 'coverage' whilst simultaneously ensuring that nodes retain line-of-sight with one another (this latter constraint arises from the need to localize the nodes; in our previous work on *mesh-based localization* [12, 13] we have shown how nodes can localize themselves in a completely unknown environment by using other nodes as landmarks). This paper describes the incremental deployment algorithm and presents the results of an extensive series of simulation experiments. These experiments serve to both validate the algorithm and illuminate its empirical properties.

1 Introduction

This paper describes a self-deployment algorithm for mobile sensor networks. A mobile sensor network is composed of a distributed collection of *nodes*, each of which has sensing, computation, communication and locomotion capabilities. It is this latter capability that distinguishes a *mobile* sensor network from its more conventional static cousins. Locomotion facilitates a number of useful network capabilities, including the ability to self-deploy and self-repair.

We envisage the use of mobile sensor networks in applications ranging from urban combat scenarios, to search-and-rescue operations and emergency environment monitoring. Consider a scenario involving a hazardous materials leak in an urban environment. Metaphorically speaking, we would like to throw a 'bucket' of sensor nodes into a building through a window or doorway. The nodes are equipped with chemical sensors that allow them to detect the relevant hazardous material. The nodes proceed to deploy themselves throughout the building in such a way that they maximize the area 'covered' by their sensors. Data from the nodes are transmitted to a base station located safely outside the building, where they are assembled to form a live map showing the concentration of hazardous compounds within the building.

For a sensor network to be useful in this scenario, the location of each node must be determined. In urban environments, it is not possible to use GPS for this purpose. Similarly, landmark-based localization approaches are generally unsuitable, since we expect that prior models of the environment are either unavailable, incomplete or inaccurate. This is particularly true in disaster scenarios, were the environment may have undergone recent (and unplanned) modifications. Fortunately, as we have recently shown [12, 13], it is possible to determine the location of nodes in a network by using the nodes themselves as landmarks; this particular technique does, however, require that nodes maintain line-of-sight with one another. Consequently, in this paper, we demand that nodes should deploy in such a way that they maximize the area 'covered' by the network, whilst simultaneously ensuring that each node can be seen by at least one other node.

The deployment algorithm described in this paper is both incremental and greedy. Nodes are deployed one-at-a-time, with each node making use of data gathered from previously deployed nodes to determine its optimal deployment location. The algorithm is *greedy* in the sense

that it attempts to determine, for each node, the location that will produce the maximum increase in the network coverage area. Unfortunately, as we show in Section 3.3, determining the 'optimal' placement (even in a greedy sense) is a fundamentally difficult problem. Consequently, the deployment algorithm described in this paper relies on a number of heuristics to guide this selection process.

We have conducted an extensive series of simulation experiments aimed at characterizing the performance of the incremental deployment algorithm. These experiments demonstrate that our algorithm, which is model free, achieves coverage results that are close to those obtained using a model-based greedy algorithm. These experiments also establish that the computation time for the algorithm is a polynomial function of order n^2 in the number of deployed nodes. We note that this algorithm has previously been demonstrated running on real hardware, in a network containing four nodes [11].

2 Related Work

Although we are not aware of any previous research that considers the specific deployment problem described here, our work is influenced and informed by a number of related problems.

The concept of *coverage* as a paradigm for evaluating many-robot systems was introduced by Gage [8]. Gage defines three basic types of coverage: blanket coverage, where the objective is to achieve a static arrangement of nodes that maximizes the total detection area; barrier coverage, where the objective is to minimize the probability of undetected penetration through the barrier; and sweep coverage, which is more-or-less equivalent to a moving barrier. According to this taxonomy, the algorithm described in this paper is a blanket coverage algorithm.

The problem of exploration and map-building by a single robot in an unknown environment has been considered by a number of authors [24, 25, 26]. The frontier-based approach of Yamauchi et al [24, 25] is particularly pertinent: this exploration algorithm proceeds by incrementally building a global occupancy map of the environment, which is then analyzed to find the 'frontiers' between free and unknown space. The robot is directed to the nearest such frontier. The network deployment algorithm described in this paper shares a number of similarities with this algorithm: we also build a global occupancy grid of the environment and direct nodes to the frontier between free and unknown space. However, in our deployment algorithm the map is built entirely from live, rather than stored, sensory data. We must also satisfy an additional constraint: that each node must be visible to at least one other node.

Multi-robot exploration and map-building has also been explored by a number of authors [4, 17, 20, 19, 3, 14] who use a variety of techniques ranging from topological matching [4] to fuzzy inference [14] and particle filters [21]. Once again, there are two key differences between these earlier works and the work described in this paper: our maps are built entirely from live, not stored, sensory data, and our deployment algorithm must satisfy an additional constraint (i.e. line-of-sight visibility). On the other hand, the heuristics used by both Simmons [19] and Burgard [3] to select goal points for exploration are strikingly similar to the heuristics used in this paper to select goal points for deployment (see Section 3.3). In effect, these heuristics state that one should not only explore the boundary of known space, but that one should also bias the exploration towards regions in which a robot is *likely* to uncover large areas of previously unknown space. Burgard describes an adaptive algorithm for making estimates of these otherwise unpredictable quantities.

The deployment problem described here is similar to that described by Bulusu et al [2], who consider the problem of adaptive beacon placement for localization in large-scale wireless sensor networks. These networks rely on RF-intensity information to determine the location of nodes; appropriate placement of RF-beacons is therefore of critical importance. The authors describe an empirical algorithm that adaptively determines the optimal beacon locations. In a somewhat similar vein, Winfield [23] considers the problem of distributed sensing in an ad-hoc wireless network. Nodes are introduced into the environment en-mass and allowed to disperse using a random-walk algorithm. Nodes are assumed to have a limited communication range, and the environment is assumed to be sufficiently large such that full network connectivity cannot be maintained. Hence the network relies on continuous random motion to bring nodes into contact, and thereby propagate information to the edges of the network. Our work differs from that described by these authors in a number of significant ways. Whereas both Bulusu and Winfield are concerned only with sensor range, we assume that network nodes are equipped with sensors that require line-of-sight to operate (such as cameras or laser range-finders). Unlike Winfield, our deployment algorithm is specifically designed to preserve network connectivity. It also aims to produce controlled *deployment*

rather than random *diffusion.* Finally, unlike Bulusu, our algorithm is *incremental* rather than *adaptive*; once nodes are deployed, they do not change location.

A mobile sensor network can also be viewed as a large-scale mobile robot formation. Such formations have been studied by a number of authors [1, 7, 18], all of whom describe methods for creating and maintaining formations via local interactions between robots. In this research, interaction with the environment is of secondary importance to interaction between the robots themselves. In contrast, the work described here emphasizes interaction with environment, and attempts to minimize interaction between network nodes.

Finally, we note that the problem of deployment is related to the traditional *art gallery* problem in computational geometry [16]. The art gallery problem seeks to determine, for some polygonal environment, the minimum number of cameras that can be placed such that the entire environment is observed. While there exist a number of algorithms designed to solve the art gallery problem, all of these assume that we possess good prior models of the environment. In contrast, we assume that prior models of the environment are either incomplete, inaccurate or non-existent. The sensor network must therefore empirically and incrementally determine the structure of the environment.

3 The Incremental Deployment Algorithm

The algorithm described here is an *incremental* deployment algorithm: nodes are deployed one at a time, with each node making use of information gathered by the previously deployed nodes to determine its ideal deployment location. The algorithm aims to maximize the total network *coverage*, i.e. the total area that can be 'seen' by the network. At the same time, the algorithm must ensure that the *visibility constraint* is satisfied; i.e. each node must be visible to at least one other node.

3.1 Assumptions, Constraints, Performance

The algorithm relies on a number of key assumptions:

- **Homogeneous nodes:** all nodes are assumed to be identical. Furthermore, we assume that each node is equipped with a range sensor (such as a laser range finder or sonar array), a broadcast communications device (such as wireless Ethernet), and is mounted on some form of mobile platform.

- **Static environment:** the environment is assumed to be static, at least to the extent that gross topology remains unchanged while the network is deploying. We assume, for example, that open doors remain open for the duration of the deployment process. Note that the deployment process *itself* will modify the environment, as nodes will both occlude and obstruct one another.

- **Model-free:** there are no prior models of the environment. This algorithm is intended for applications in which environment models are unavailable, incomplete or inaccurate. Indeed, a key task for the network may be to *generate* such models.

- **Localization:** the pose of each and every node is known in some arbitrary global coordinate system.

In our previous work on *mesh-based localization* [12, 13], we have shown how global localization can be performed using only the measured relationships between network nodes. This technique does not require external landmarks or prior models of the environment. It does, however, require that each node be visible to at least one other node. It is this requirement gives rise to the visibility constraint, which we define as follows:

- **Visibility:** each node must be visible to at least one other node at its deployed location.

This constraint does not necessarily imply that nodes must be visible whilst they are in motion; we assume that nodes are equipped with some form of odometry or inertial navigation that allows them to navigate during these periods.

We evaluate the incremental deployment algorithm using two performance metrics:

- **Coverage:** the total area visible to the network's sensors.

- **Time:** the total deployment time, which includes both the time taken to perform the necessary computations (CPU time) and the time taken to physically move the nodes (real time).

Naturally, we wish to maximize coverage whist minimizing the deployment time.

3.2 Algorithm Overview

The incremental deployment algorithm has four phases: initialization, selection, assignment and execution.

- **Initialization.** Nodes are assigned one of three states: *waiting*, *active* or *deployed*. As the names suggest, a waiting node is waiting to be deployed, an active node is in the process of deploying, and a deployed node has already been deployed. Initially, the state of all nodes is set to waiting, with the exception of a single node that is set to deployed. This node provides a starting point, or 'anchor', for the network, and is not subject to the visibility constraint.
- **Selection.** Sensor data from the deployed nodes is combined to form a common map of the environment. This map is analyzed to select the deployment location, or goal, for the next node.
- **Assignment.** In the simplest case, the selected goal is assigned to the first waiting node, and the node's state is changed from waiting to active. Assignment is complicated by the fact that deployed nodes tend to obstruct the passage of waiting nodes, necessitating a more complex assignment algorithm. This algorithm may have to re-assign the goals of any number of deployed nodes, changing their state from deployed to active.
- **Execution.** Active nodes are deployed sequentially to their goal locations. The state of each node is changed from active to deployed upon arrival at the goal.

The algorithm iterates through the selection, assignment and execution phases, terminating only when all nodes have been deployed.

3.3 Selection

The selection phase determines the next deployment location, or goal. Ideally, this goal should maximize the coverage metric whilst simultaneously satisfying the visibility constraint. Unfortunately, there is no way of determining the 'optimal' goal a priori, not even in a greedy or local sense. Since we lack a prior model, and must instead rely on sensed data from deployed nodes, our knowledge of and reasoning about the environment is necessarily incomplete. For this reason, the algorithm described here avoids such reasoning altogether. Instead, we use a number of relatively simple goal selection *policies* that rely on heuristics to guide the selection process.

As a first step, sensor data from the deployed nodes are combined to form an *occupancy grid* [5, 6]. Each cell in this grid is assigned one of three states: *free*, *occupied* or *unknown*. A cell is *free* if it is known to be contain no obstacles, *occupied* if it is known to contain a one or more obstacles, and *unknown* otherwise. We use a standard Bayesian technique [6] to determine the *probability* that each cell is occupied, then threshold this probability to determine the state of each cell.

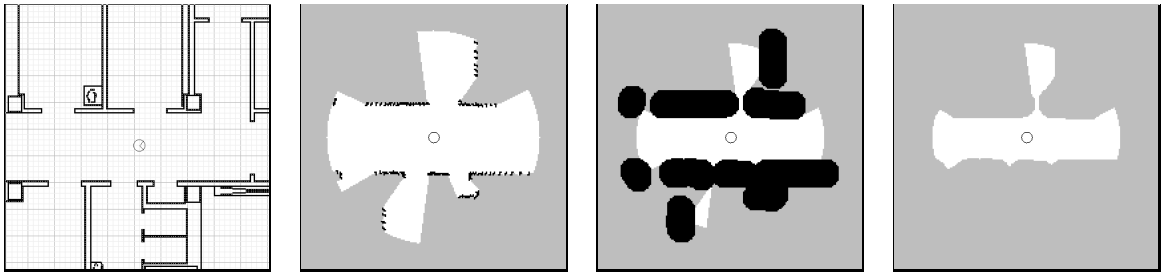
Any cell that can be seen by one or more nodes will be marked as either free or occupied; only those cells that cannot be seen by *any* node will marked as unknown. [1] Therefore, we can ensure that the visibility constraint is satisfied by always selecting goals that lie somewhere in free space. Unfortunately, not all free space cells represent valid deployment locations. Since nodes have finite size, a free cell that is close an occupied cell may not be reachable. Similarly, we must eliminate free cells that are close to unknown cells, since these unknown cells may also turn out to be occupied. There may also exist free cells that are far from both occupied and unknown cells, but are nevertheless unreachable: a node may, for example, be able to see through an opening that is too narrow to allow passage. To simplify this kind of analysis, we post-process the occupancy grid to form both a *configuration grid* and a *reachability grid*.

As the name suggests, a configuration grid is a representation of the nodes' *configuration space* [15]. Each cell in the configuration grid can have one of three states: *free*, *occupied* and *unknown*. A cell is *free* if and only if all the occupancy grid cells lying within a certain distance d are also free (the distance d is usually set to a value greater than or equal to the node's radius). A cell is *occupied* if there are one or more the occupancy grid cells lying within distance d that are similarly occupied. All other cells are marked as *unknown*.

A node can be safely placed at any free cell in the configuration grid. To determine whether such a cell is also *reachable*, we further process the configuration grid to derive the reachability grid. This is done by applying a flood-fill algorithm to free space in the configuration grid, starting from the location of each deployed node in turn. Cells in the reachability grid are thus labeled as either *reachable* or *unreachable*.

Figure 1 shows an example of the occupancy, configuration and reachability grids generated for a single node

[1] Strictly speaking, since simple Bayesian reasoning does not distinguish between ignorance and contradiction, a cell may also be marked as unknown if there is contradictory evidence regarding its occupancy state.



(a) (b) (c) (d)

Figure 1: (a) A fragment of the simulated environment containing a single node. (b) Occupancy grid: black cells are occupied, white cells are free, gray cells are unknown. (c) Configuration grid: black cells are occupied, white cells are free, gray cells are unknown. (d) Reachability grid: white cells are reachable, gray cells are unreachable.

in a simulated environment. Note that the set of reachable cells is a subset of the set of free configuration cells, which is in turn a subset of the set of free occupancy cells. Thus, by selecting a goal that lies within a reachable cell, we simultaneously ensure that the deploying node will be visible, that it will not be in collision, and that there exists some path such that the node can reach the goal.

Having determined the reachability space, the selection algorithm makes use of two heuristics to guide final goal selection: a *boundary* heuristic and a *coverage* heuristic.

- **Boundary heuristic:** nodes should deploy to the boundary between free and unknown space.

This heuristic seeks to place nodes in such a way that there is minimal overlap between sensory fields, thereby maximizing the coverage metric.

- **Coverage heuristic:** nodes should deploy to the location at which they will 'cover' the greatest area of presently unknown space.

This heuristic seeks to place nodes at the location at which they have the greatest potential to increase the coverage area, given that we make the optimistic assumption that all unknown areas are, in fact, free space. There is no guarantee that this assumption is correct, of course; the node may deploy to a location that appears to cover a large area of unknown space, only to find that it has deployed itself into a closet.

In and of themselves, the heuristics do not necessarily specify a unique goal. They can, however, be incorporated into a number of goal selection *policies*, each of which will fully determine a unique goal. We have implemented four such selection policies:

- **P1:** randomly select a location in free space.
- **P2:** randomly select a location on the free/unknown boundary.
- **P3:** select the free space location that maximizes the coverage heuristic.
- **P4:** select the free/unknown boundary location that maximizes the coverage heuristic.

These policies express all possible combinations of the two heuristics, including the 'control' case in which neither heuristic is used. The first two are stochastic, while the latter two are deterministic. Note that P4 is a special case of P3; it is included partly for completeness, and partly because it can be computed much more rapidly than P3. In Section 4, we will compare the performance of these four policies in an experimental context, and attempt to determine the relative contributions of the underlying heuristics.

3.4 Assignment

The assignment phase attempts to assign the newly selected goal to a waiting node. This process is complicated by the fact that nodes may find themselves unable to reach some parts of the environment due to obstruction from previously deployed nodes. Such obstruction becomes increasingly likely as the size of the nodes approaches the size of openings in the environment. There is, fortunately, a very natural solution to this problem that exploits the homogeneity of the network nodes: an obstructed node may swap goals with the node obstructing it. Thus, if node A is obstructed by node B, node B can move to A's

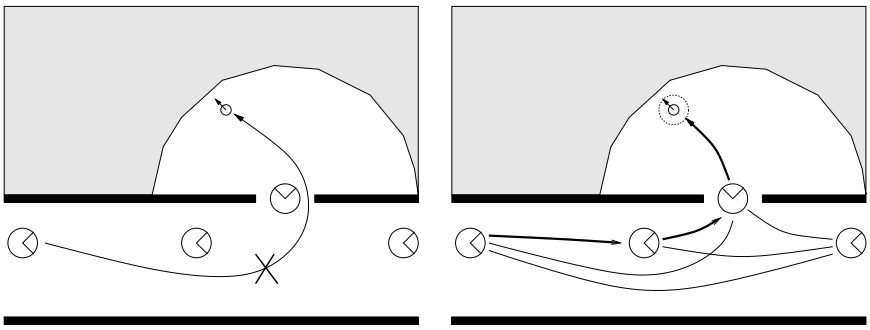


Figure 2: (a) A typical obstruction problem, with a waiting node unable to reach its deployment location. The gray area indicates the region of space that is not yet covered by the network. (b) The obstruction is resolved by re-assigning the deployment location to another node.

deployment location, while A replaces B at its original deployment location. Since all nodes are assumed to be equivalent, this goal-swapping makes no functional difference to the network. For complex environments, with many obstructions, this resolution strategy may need to be applied recursively: A replaces B, B replaces C, C replaces D and so on.

The assignment phase uses a slightly modified version of this procedure, in which we do not attempt to directly infer which nodes are obstructing which other nodes. The algorithm is as follows.

- Construct a graph in which each vertex represents a network node and each edge represents a reachability relationship between two nodes (i.e. node A can reach node B's position, and vice-verse). The length of each edge corresponds to the distance between the nodes, and the goal is represented by a dummy vertex.
- Find the shortest path from the first waiting node to to goal. The length of any path through the graph is given by the sum of edge lengths, and the shortest path is found using dynamic programming.
- Mark every node on the shortest path as active, and assign each node the goal of reaching the position currently occupied by the next node along the path.

This algorithm is illustrated in Figure 2, which shows a proto-typical graph with the shortest path highlighted. Note that while it is not strictly necessary for all of the nodes on this path to move, all of the potential obstructions have been resolved.

The assignment algorithm requires that we determine the reachability relationship and distance between $n^2/2$ pairs of nodes. In principle, this requires that we generate a *plan* for reaching every node from every other node. In practice, we can simplify this process by generating a unique *distance transform* [26] for each node. The distance transform is a simple form of dynamic programming: distances are propagated out from the node, traveling through free configuration space and around occupied or unknown space. Ultimately, a distance will be assigned to each cell from which the node can be reached. The graph is constructed by simply reading off these distances.

The assignment algorithm described above produces some interesting behavior: the network will tend to 'ooze' out from its starting location, with many nodes being active at any given point in time. In addition, as the network spreads through the environment, the same nodes will tend to remain on edge of the network. Note that there are many, many alternative assignment algorithms that we could choose, some of which will produce radically different network behavior. In this paper, however, we will consider only the simple algorithm described above.

3.5 Execution

During the execution phase, active nodes are deployed to their goal locations. Nodes are deployed using sequential execution; i.e. we wait for each node to reach its goal before deploying the next node. Active nodes are deployed in the order in which they were assigned goals: the first node will move to the new deployment location, the sec-

ond will move to take up the first node's old location, and so on. Since there is only one node in motion at any given point in time, and since the goal resolution algorithm ensures that each successive goal is unobstructed, there is no possibility for interference between nodes.

Sequential execution is, however, quite slow: execution time is proportional to the sum of the distances traveled by the active nodes, which is, in turn, equal to the distance a single node would have to travel if there were no obstructions. As the network becomes bigger, nodes will have farther to travel, and hence we expect that execution time will increase more-or-less linearly with network size. There are alternatives to sequential execution: if we assume that nodes are equipped with some mechanism for resolving interference, we can use *concurrent* execution, in which all active nodes are set in motion at the same time. This also has implications for the assignment phase of the algorithm, which must be appropriately modified to make full use of concurrent execution. Through such modifications, it is possible, in principle, to create an algorithm in which execution time is constant, irrespective of network size. This topic is, unfortunately, beyond the scope of this paper.

4 Experiments and Analysis

We have conducted a series of simulation experiments aimed at determining the empirical properties of the incremental deployment algorithm. Two metrics are of particular interest: coverage (how much of the environment does the network cover), and time (how long does the network take to deploy). In both cases, we are interested not only in the properties of the 50-node network used in these experiments, but also in the scaling properties of the algorithm. That is, we would like to understand the consequences of increasing the network size into the range of hundreds or thousands of nodes.

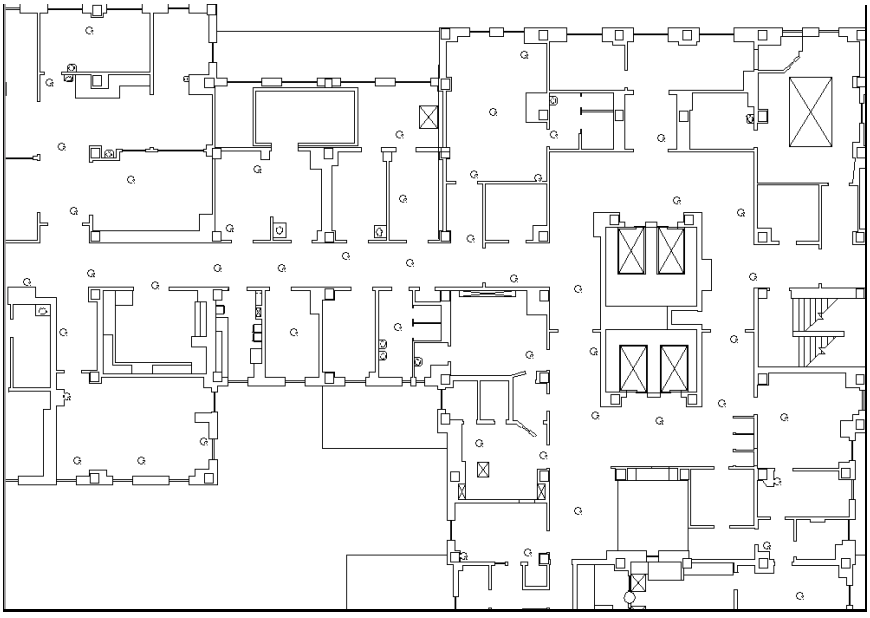
Our experiments were conducted using the Player robot server [10] in combination with the Stage [22, 9] multi-agent simulator. Stage simulates the behavior of real sensors and actuators with a high degree of fidelity; algorithms developed using Stage can usually be transferred to real hardware with little or no modification. The sensor network for these experiment consists of 50 nodes, each of which is equipped with a scanning laser range finder with a 360 degree field-of-view mounted on a differential mobile robot base. Each node is also equipped with an 'ideal' localization sensor that provides accurate position and orientation information. This sensor is used in place of the mesh-based localization technique described in [12, 13], as this technique has not yet been merged with the incremental deployment algorithm. The simulated nodes were placed in the environment shown in Figure 3. This is a fragment of a much larger environment that represents a single floor in a large hospital.

We conducted a large set of trials, varying for each trial the selection policy, starting location and sensor range. Starting locations were chosen from a set of 10 pre-selected locations. Sensor range was taken to be 2, 4, 6 or 8m. For the stochastic policies P1 and P2, 10 trials where conducted for each combination of initial location and sensor range (a total of 400 trials for each policy). For the deterministic policies P3 and P4, a single trial was conducted from each combination of initial location and sensor range (a total of 40 trials for each policy). In each trial, we measured network coverage, computation and execution time.

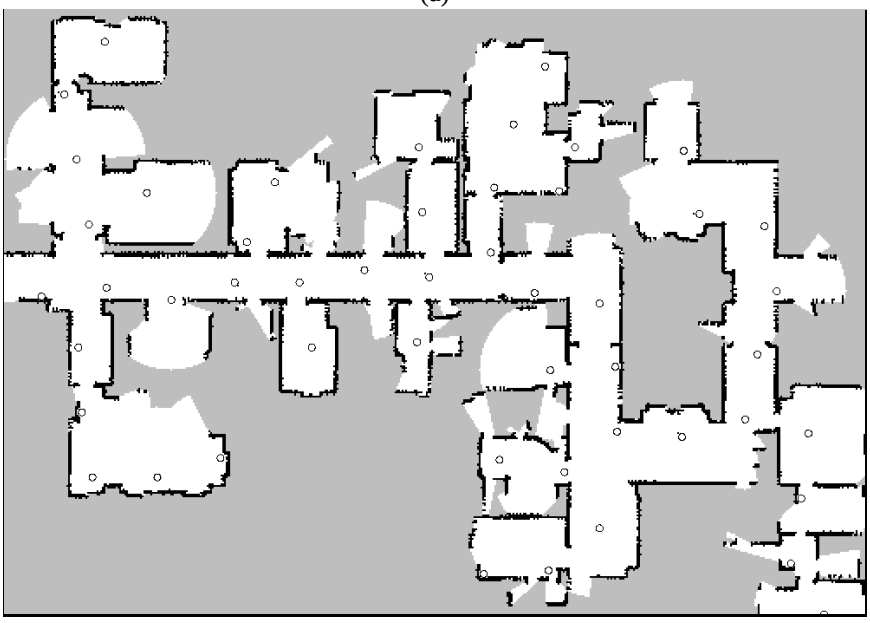
4.1 Coverage

Figure 4 shows a plot of network coverage as a function of the number of deployed nodes. Coverage is measured by counting the number of free cells in the occupancy grid and multiplying by the area covered by each cell. The figure shows the results for each policy, averaged over all initial locations; the sensor range is 4m. Variance is indicated by the error bars (most of which have been omitted for clarity). Inspecting these plots, it is apparent that coverage increases linearly with the number of deployed nodes, irrespective of the selection policy. It is also clear that the selection policies P2 to P4, which make use of the heuristics described in Section 3.3, perform significantly better than policy P1, which is the control case (i.e. random deployment).

We can make this comparison more precise by defining, for each policy, a *coverage factor* α that measures the average area 'covered' by each node. That is, α is such that the total network coverage is approximately equal to $\alpha n + \beta$, where n is the number of deployed nodes and β is some constant. Table 1 lists the coverage factors for sixteen different combinations of selection policy and sensor range (determined using simple linear regression). It should be noted that these values are meaningful only when the total coverage area is much less than the total area of the environment. In any bounded environment, network coverage must eventually saturate, and boundary effects are likely to introduce significant non-linearities. In our experiments, the environment is very large and boundary effects have minimal impact (although



(a)



(b)

Figure 3: (a) A fragment of the simulated environment. (b) Occupancy grid produced by a typical deployment (policy P4 with a sensor range of 4m).

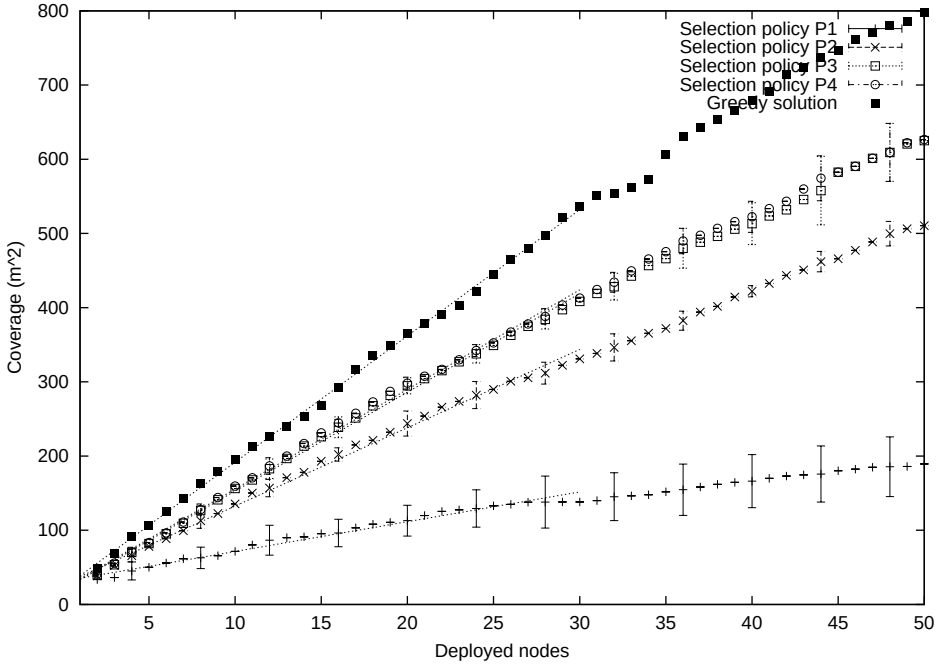


Figure 4: Network coverage for selection policies P1 to P4; the sensor range is 4m. Note that most of the error bars have been suppressed for the sake of clarity.

one can possibly see the start of such effects in some of the coverage plots in Figure 4).

Inspecting the values in Table 1, it is apparent that the three goal selection policies that incorporate one or more of the heuristics described in Section 3.3 (policies P2 to P4) perform significantly better than the control case (policy P1). Policies P3 and P4, in fact, produce an almost 3-fold improvement over simple random deployment. It is also apparent that most of this improvement can be achieved using the boundary heuristic alone: policy P2 (which uses only the boundary heuristic) is almost as good as policy P3 (which uses only the coverage heuristic). Furthermore, policies P3 and P4 are almost indistinguishable, suggesting that the coverage heuristic will, in almost all situations, deploy nodes to the free/unknown boundary. Thus, it makes sense to use policy P4 in preference to P3, since the latter requires much more time to compute and produces negligible improvement in network coverage (we will look at exactly how much more time P3 requires in the next section).

Comparing the coverage factors obtained using different sensor ranges is also illuminating: not so much for what it tells us about the algorithm, but for what it tells us about the environment. Naively, one would expect network coverage to increase as the square of the sensor range, since doubling the range of a single sensor will quadruple its coverage area. In a real environment, of course, things are not quite so simple: occlusion, not sensor range, will dominate the placement of nodes. Inspecting Table 1, we can see that there is significant improvement in coverage as one increases sensor range from 2m to 6m, but minimal improvement thereafter. This is true for all four selection policies. For this environment, 6m appears to be a 'characteristic length'; it may be, for example, that this distance corresponds to the average distance between doorways, or to the average size of a room. It would be interesting to conduct further experiments in different environments, in an attempt to correlate the coverage factors with environment structure.

Ideally, we would like to compare these coverage results against the *optimal* value, i.e. the greatest possible coverage that can be obtained for a network that satisfies the visibility constraint. Naturally, when determining the optimal coverage, we assume that we have a perfect a priori model of the environment. Even so, determining the optimal coverage is extremely difficult, since it necessi-

Policy	Range 2m	4m	6m	8m
P1	1.50 ± 0.07	4.01 ± 0.20	6.07 ± 0.42	7.48 ± 0.39
P2	3.63 ± 0.04	10.56 ± 0.13	14.30 ± 0.31	15.68 ± 0.40
P3	4.86 ± 0.05	13.31 ± 0.11	18.40 ± 0.38	19.33 ± 0.48
P4	4.86 ± 0.05	13.42 ± 0.09	18.20 ± 0.38	19.30 ± 0.48
Greedy	5.71 ± 0.03	17.01 ± 0.11	24.65 ± 0.44	27.14 ± 0.88

Table 1: Coverage factors for selection policies P1 to P4 and sensor ranges 2, 4, 6 and 8m.

tates a search over the space the space of all possible networks. This space is vast. Consider a network of n nodes in an environment of area A. If we discretize this environment into locations that are distance D apart, the total number of possible networks is $\frac{(A/D^2)!}{n!(A/D^2-n)!}$ (not all of which will satisfy the visibility constraint, of course). For a relatively small network with $n = 10$, $A = 100m^2$ and $D = 0.1m$, the number of possible networks is around 10^{40}. Clearly, a brute force search of this space is impractical. While there may exist closed form solutions or good approximations for this problem (it is, for example, similar to the art gallery problem [16]), we are not aware of any such solutions at this time.

Instead of comparing our results with the optimal solution, we will instead compare them with the best *greedy* solution. The greedy solution is obtained by constructing the network incrementally, choosing for each node the location that produces the greatest coverage. For our algorithm, the greedy solution is a fairer test than the optimal solution, since it represents the best result that can be expected for any form of *incremental* deployment algorithm.

In practice, we generate the greedy solution using the simulator and a modified form of the incremental deployment algorithm. For each node, we first compute the reachability grid, then 'teleport' the node to every reachable cell in succession. At each location, we measure the network coverage. Finally, the node is teleported back to the location that produces the greatest coverage, and the process is repeated for the next node.

Table 1 shows the coverage factors for the greedy solution. Note that the factors for policies P3 and P4 are within 70% to 80% of the greedy values: this suggests that our heuristics are very good indeed, and that our policies are about as good as they are likely to get for a model-free algorithm.

4.2 Time

We will consider separately the temporal properties of each of the three main phases of the algorithm: selection, assignment and execution. In the case of selection and assignment, we are interested in the time spent in computation; in the case of execution, we are interested in the time spent moving nodes (wall-time).

4.2.1 Selection

Figure 5 shows the measured computation time for the selection phase of the algorithm, plotted against the number of deployed nodes (note that this is a log-log scale). The four selection policies are plotted separately, with each plot representing an average over all initial locations. The sensor range in all cases is 4m.

Note that all four plots become linear as the number of deployed nodes n increases: this implies that computation time is a polynomial function of the number of deployed nodes. If we assume that this function has a high-order term of the form bn^a, we can characterize each policy in terms of its exponent a and coefficient b. Table 2 lists the a and b values for polices P1 to P4. These values were calculated using linear regression in log-log space, using only the last 30 data points for each policy (since we are trying to capture the highest-order term only).

Inspecting this table, two results are immediately apparent. First, and most important, selection time scales sub-linearly with the number of deployed nodes (the exponent a for all policies is less than 1). This result conforms only partially to our theoretical expectations. The selection phase of the algorithm can be broken into two parts: map generation and policy application. For map generation, data from each node are added to the occupancy grid sequentially and independently; hence we expect map generation to scale linearly. For policy application, the computation time is dependent on the particular selection policy used: for policies using the bound-

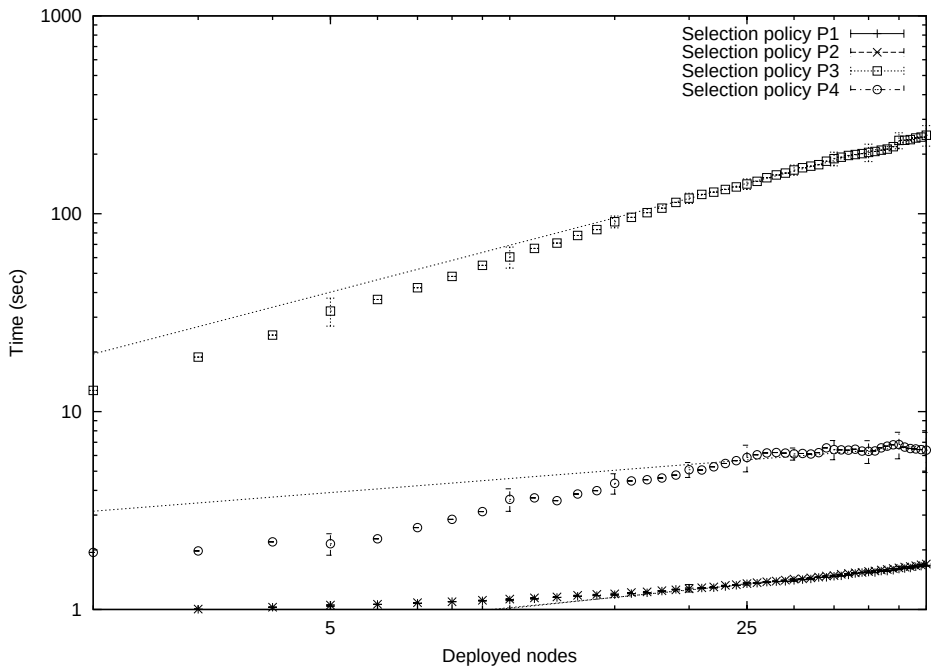


Figure 5: Selection time (CPU) for policies P1 to P4. The scale is log-log. Most of the error bars have been suppressed for the sake of clarity.

ary heuristic, computation time will be proportional to the free/unknown boundary length; for policies using the coverage heuristic, computation time will be proportional to the free space area. If we assume that both boundary length and free space area are proportional to the number of deployed nodes, computation time for policy application will also scale linearly. We attribute the sub-linear results in Table 2 to a combination of two factors: selection time is dominated by policy application rather than map generation, and our assumption that boundary length scales linearly with the number of deployed nodes is most probably incorrect. If we were to increase the number of nodes in these experiments, we expect that map generation would ultimately dominate, and that a would subsequently approach 1.

The second result to note from Table 2 is that policy P4, which is almost indistinguishable from P3 in terms of coverage, is about 4 times faster (consider the coefficient b); this confirms our earlier conclusion that P4 should, in general, be used in preference to P3.

4.2.2 Assignment

Figure 6 shows the measured computation time for the assignment phase of the algorithm (on a log-log plot). The four selection policies are plotted separately, with each plot representing an average over all initial locations; the sensor range in all cases is 4m. These plots are clearly linear, suggesting that computation time for the selection phase is a polynomial function of the number of deployed nodes. Table 2 lists the a and b values for the selection phase: this phase clearly scales as n^2 in the number of deployed nodes n.

The scaling properties of the assignment phase conform exactly to our theoretical expectations. During this phase, we generate n separate distance transforms, the computation time for each of which scales linearly with the free space area. Since the free space area also scales linearly with n (as we showed in Section 4.1), the assignment phase will necessarily scale as $n \times n = n^2$.

Ideally, we would like this phase of the algorithm to scale linearly or better. We are actively seeking alternative algorithms with this property.

	Selection		Assignment		Execution	
Policy	a	b	a	b	a	b
P1	0.30 ± 0.00	0.52 ± 0.01	1.82 ± 0.02	0.01 ± 0.00	0.91 ± 0.09	0.77 ± 0.24
P2	0.31 ± 0.00	0.50 ± 0.01	1.80 ± 0.01	0.01 ± 0.00	0.77 ± 0.08	2.05 ± 0.60
P3	0.79 ± 0.02	11.33 ± 0.92	1.86 ± 0.01	0.01 ± 0.00	0.51 ± 0.10	2.22 ± 0.79
P4	0.24 ± 0.04	2.66 ± 0.43	1.86 ± 0.01	0.01 ± 0.00	0.50 ± 0.11	2.41 ± 0.98

Table 2: Time constants for the three phases of the algorithm. Time is assumed to be a polynomial function of the number of deployed nodes n, with a high-order term of the form bn^a.

4.2.3 Execution

Figure 7 shows the wall-clock time (i.e. the elapsed real-time, not CPU time) for the execution phase of the algorithm (plotted on a log-log scale). The four selection policies are plotted separately, with each plot representing an average over all initial locations. The sensor range in all cases is 4m. While there is clearly a great deal of variance in the deployment time, the general trend in all four plots is clearly linear, suggesting once again that execution time is a polynomial function of the number of deployed nodes. Table 2 lists the a and b values for the execution phase; inspecting the a values, it is apparent that while the execution time for policy P1 scales more-or-less linearly with the number of deployed nodes n, the remaining policies scale sub-linearly.

These results are intriguing, but not entirely unexpected. With sequential deployment, execution time is proportional to the sum of the distances traveled by the active nodes, which is, in turn, equal to the distance that would be traveled by a single node in an obstruction-free environment. For the random deployment policy P1, we expect that this distance will scale linearly with the free space area and hence with the number of deployed nodes. For the remaining selection policies, which seek to place nodes on the free-space boundary (either explicitly, as in the case of P2 and P4, or implicitly, as in the case of P3), the scaling properties will depend on the nature of the environment. If, for example, the environment consists of a single corridor which can only fit one node abreast, the distance to the boundary will scale linearly with the free space area. If, on the other hand, the environment is completely empty, the distance traveled will scale as the *square-root* of the free space area. The results in Table 2 suggest that, for policies P3 and P4, this environment is effectively 'empty' (i.e. these policies scale as $n^{\frac{1}{2}}$). For policy P1, on the other hand, the environment is only partially empty.

Note that we ideally like execution time to be constant rather than linear or $n^{\frac{1}{2}}$. Consider the network coverage *rate*, i.e. the change in coverage as a function of wall-time. If execution time is linear, this rate will necessarily decrease as the number of deployed nodes grows. Consequently, rather than increasing linearly with time, network coverage will increase only logarithmically. Linear growth can only be achieved if execution time is constant, which implies that some form of concurrent execution must be used (i.e. many nodes must move at the same time). As noted in Section 3.5, concurrent execution requires a more advanced assignment algorithm, together with some form of interference resolution strategy. While we are actively researching these topics, they are, unfortunately, beyond the scope of this paper.

5 Conclusion and Further Work

The experiments described in Section 4 clearly establish the utility of the incremental deployment algorithm and the heuristics on which it is based. The coverage results for policies P3 and P4 (which are model free) are between 70% and 85% of the results obtained for a model-based greedy algorithm. Furthermore, the algorithm scales as a polynomial function of the number of deployed nodes, and is in the worst case of order n^2. On a practical note, we have also demonstrated that the algorithm can handle a large number of nodes (50) using modest computational resources (our simulations were performed in real-time on a single workstation).

The key weakness of these experiments is, perhaps, their reliance on a global localization mechanism other than the mesh-based method for which the incremental deployment algorithm was designed (the visibility constraint arises directly from the need of this latter method to maintain line-of-sight relationships between nodes). While we have previously demonstrated mesh-based localization for mobile sensor networks [12, 13], this method is not yet integrated with the incremental

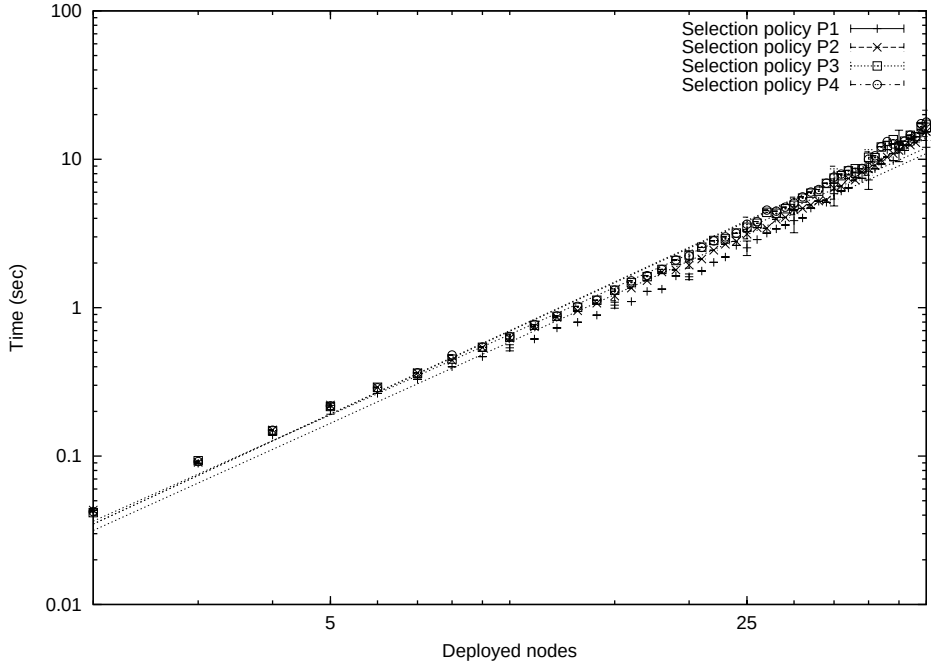


Figure 6: Resolution time (CPU) for policies P1 to P4. The scale is log-log. Most of the error bars have been suppressed for the sake of clarity.

deployment algorithm described here. We are currently performing this integration, and expect to demonstrate a combined system in the very near future.

We have already taken the first steps to demonstrating this algorithm running on real hardware in a real environment. The algorithm has been implemented and tested on a four-node network in a controlled environment [11]; we are currently preparing a much more ambitious experiment involving up to 9 nodes in an unmodified environment. Thus we expect to demonstrate the utility of the incremental deployment algorithm for real applications in real environments.

References

[1] T. Balch and M. Hybinette. Behavior-based coordination of large-scale robot formations. In *Proceedings of the Fourth International Conference on Multiagent Systems (ICMAS '00)*, pages 363–364, Boston, MA, USA, July 2000.

[2] N. Bulusu, J. Heidemann, and D. Estrin. Adaptive beacon placement. In *Proceedings of the Twenty First International Conference on Distributed Computing Systems (ICDCS-21)*, Pheonix, Arizona, April 2001.

[3] W. Burgard, M. Moors, D. Fox, R. Simmons, and S. Thrun. Collaborative multi-robot exploration. In *Proc. of IEEE International Conferenceon Robotics and Automation (ICRA)*, volume 1, pages 476–81, 2000.

[4] G. Dedeoglu and G. S. Sukhatme. Landmark-based matching algorithms for cooperative mapping by autonomous robots. In L. E. Parker, G. W. Bekey, and J. Barhen, editors, *Distributed Autonomous Robotics Systems*, volume 4, pages 251–260. Springer, 2000.

[5] A. Elfes. Sonar-based real-world mapping and navigation. *IEEE Journal of Robotics and Automation*, RA-3(3):249–265, 1987.

[6] A. Elfes. Occupancy grids: A stochastic spatial representation for active robot perception. In *Proceedings of the Sixth Conference on Uncertainty in AI*. Morgan Kaufmann Publishers, Inc, July 1990.

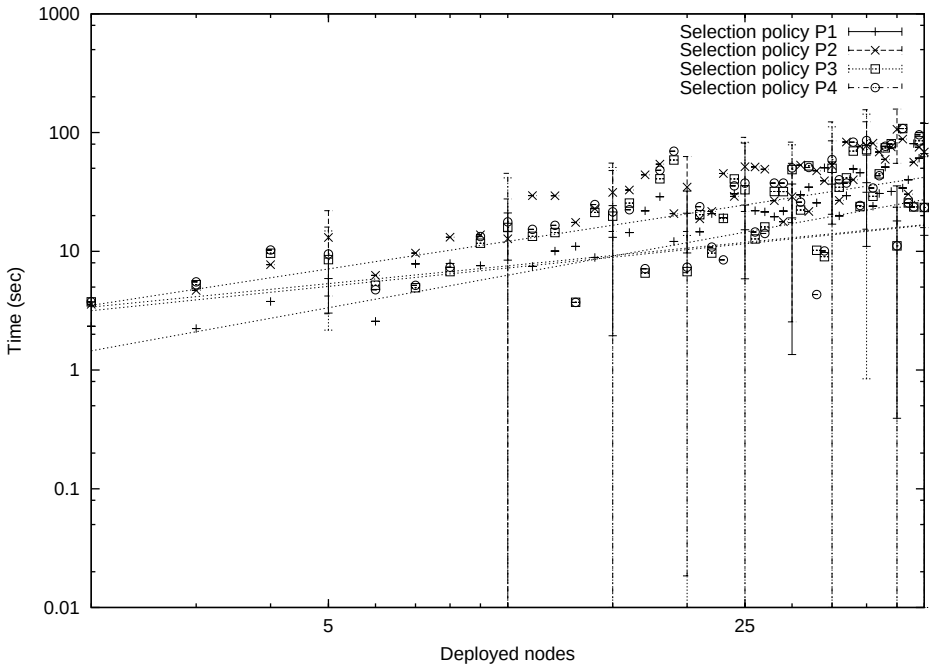


Figure 7: Execution time (wall-clock) for policies P1 to P4. The scale is log-log. Most of the error bars have been suppressed for the sake of clarity.

[7] J. Fredslund and M. J. Matarić. Robot formations using only local sensing and control. In *International Symposium on Computational Intelligence in Robotics and Automation (IEEE CIRA 2001)*, Banff, Alberta, Canada, July 2001.

[8] D. W. Gage. Command control for many-robot systems. In *AUVS-92, the Nineteenth Annual AUVS Technical Symposium*, pages 22–24, Hunstville Alabama, USA, June 1992. Reprinted in Unmanned Systems Magazine, Fall 1992, Volume 10, Number 4, pp 28-34.

[9] B. Gerkey, R. Vaughan, and A. Howard. Player/Stage homepage. http://www-robotics.usc.edu/player/, September 2001.

[10] B. P. Gerkey, R. T. Vaughan, K. Støy, A. Howard, G. S. Sukhatme, and M. J. Matarić. Most valuable player: A robot device server for distributed control. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, page to appear, Wailea, Hawaii, Oct. 2001.

[11] A. Howard and M. J. Matarić. Cover me! a self-deployment algorithm for mobile sensor networks. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, page (submitted), 2002.

[12] A. Howard, M. J. Matarić, and G. S. Sukhatme. Relaxation on a mesh: a formalism for generalized localization. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, page (to appear), 2001.

[13] A. Howard, M. J. Matarić, and G. S. Sukhatme. Self-localization in a distributed sensor network. Technical Report IRIS-01-407, Institute for Robotics and Intelligent Systems Technical Report, University of Sourthern California, 2001.

[14] M. López-Sánchez, F. Esteva, R. L. de Mántaras, C. Sierra, and J. Amat. Map generation by cooperative low-cost robots in structured unknown environments. *Autonomous Robots*, 5(1):53–61, 1998.

[15] T. Lozano-Perez and M. Mason. Automatic synthesis of fine-motion strategies for robots. *International Journal of Robotics Research*, 3(1):3–24, 1984.

[16] J. O'Rourke. *Art Gallery Theorems and Algorithms*. Oxford University Press, New York, August 1987.

[17] I. M. Rekleitis, G. Dudek, and E. E. Milios. Graph-based exploration using multiple robots. In L. E. Parker, G. W. Bekey, and J. Barhen, editors, *Distributed Autonomous Robotics Systems*, volume 4, pages 241–250. Springer, 2000.

[18] F. E. Scheider, D. Wildermuth, and H.-L. Wolf. Motion coordination in formations of multiple mobile robots using a potential field approach. In L. E. Parker, G. W. Bekey, and J. Barhen, editors, *Distributed Autonomous Robotics Systems*, volume 4, pages 305–314. Springer, 2000.

[19] R. Simmons, D. Apfelbaum, W. Burgard, D. Fox, M. Moors, S. Thrun, and H. Younes. Coordination for multi-robot exploration and mapping. In *Proc. of the Seventeenth National Conference on Artificial Intelligence (AAAI-2000)*, pages 852–858, 2000.

[20] S. Thrun, W. Burgard, and D. Fox. A real-time algorithm for mobile robot mapping with applications to multi-robot and 3d mapping. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA2000)*, volume 1, pages 321–328, 2000.

[21] S. Thrun, D. Fox, W. Burgard, and F. Dellaert. Robust monte carlo localization for mobile robots. *Artificial Intelligence Journal*, 128(1–2):99–141, 2001.

[22] R. T. Vaughan. Stage: a multiple robot simulator. Technical Report IRIS-00-393, Institute for Robotics and Intelligent Systems, University of Southern California, 2000.

[23] A. F. Winfield. Distributed sensing and data collection via broken ad hoc wireless connected networks of mobile robots. In L. E. Parker, G. W. Bekey, and J. Barhen, editors, *Distributed Autonomous Robotics Systems*, volume 4, pages 273–282. Springer, 2000.

[24] B. Yamauchi. Frontier-based approach for autonomous exploration. In *Proceedings of the IEEE International Symposium on Computational Intelligence, Robotics and Automation*, pages 146–151, 1997.

[25] B. Yamauchi, A. Shultz, and W. Adams. Mobile robot exploration and map-building with continuous localization. In *Proceedings of the 1998 IEEE/RSJ International Conference on Robotics and Automation*, volume 4, pages 3175–3720, 1998.

[26] A. Zelinksy. A mobile robot exploration algorithm. *IEEE Transactions on Robotics and Automation*, 8(2):707–717, 1992.