Autonomous Robots, Special Issue on Intelligent Embedded Systems, 13(2), Sept 2002, pp. 113--126

An Incremental Self-Deployment Algorithm for Mobile Sensor Networks

Andrew Howard, Maja J Matarić and Gaurav S Sukhatme
Robotics Research Laboratory, Computer Science Department, University of Southern California, Los Angeles, California 90089-0781, U.S.A
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 described in this paper will deploy such nodes one-at-a-time into an unknown environment, with each node making use of information gathered by previously deployed nodes to determine its deployment location. The algorithm is designed to maximize network 'coverage' while simultaneously ensuring that nodes retain line-of-sight relationships with one another. This latter constraint arises from the need to localize the nodes in an unknown environment: in our previous work on *team localization* (Howard et al., 2002b) we have shown how nodes can localize themselves by using *other nodes* as landmarks. This paper describes the incremental deployment algorithm and presents the results from an extensive series of simulation experiments. These experiments serve to both validate the algorithm and illuminate its empirical properties.

Keywords: sensor networks, deployment, multi-robots systems.

1. Introduction

This paper describes an incremental 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, for example, 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, and deploy themselves throughout the building in such a way that they maximize the area 'covered' by these 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 the sensor network to be useful in this scenario, the location of each node must be determined. In urban environments, accurate localization using GPS is generally not possible (due occlusions or multi-path effects), while landmark-based approaches require prior models of the environment that may be either unavailable, incomplete or inaccurate. This is particularly true in disaster scenarios, were the environment may have undergone recent (and unplanned) structural modifications. Fortunately, as we have recently shown (Howard et al., 2002b; Howard et al., 2002d), it is possible to determine the location of network nodes by using the nodes *themselves* as landmarks. This particular technique does, however, require that nodes maintain line-of-sight relationships 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, while 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. Unfortunately, as we show in Section 3.2, determining the optimal placement (even in a greedy sense) is a fundamentally difficult problem. The deployment algorithm described in this paper therefore relies on a number of heuristics to guide the selection of deployment locations.

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.

2. Related Work

The concept of *coverage* as a paradigm for evaluating *multi-robot* systems was introduced by (Gage, 1992). 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 (Yamauchi, 1997; Yamauchi et al., 1998; Zelinksy, 1992). The frontier-based approach described in (Yamauchi, 1997; Yamauchi et al., 1998) 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 Yamauchi's 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 been explored by a number of authors (Dedeoglu and Sukhatme, 2000; Rekleitis et al., 2000; Thrun et al., 2000; Simmons et al., 2000; Burgard et al., 2000; López-Sánchez et al., 1998) who use a variety of techniques ranging from topological matching (Dedeoglu and Sukhatme, 2000) to fuzzy inference (López-Sánchez et al., 1998) and particle filters (Thrun et al., 2001). 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 et al., 2000) and (Burgard et al., 2000) to select goal points for exploration are strikingly similar to the heuristics used in this paper to select goal points

for deployment (see Section 3.2). 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.

A distributed algorithm for the deployment of mobile robot teams has been described by (Payton et al., 2001). Payton introduces the concept of 'virtual pheromones': localized messages that are emitted by one robot and detected by nearby robots. Virtual pheromones can be used to generate either 'gas expansion' or 'guided growth' deployment models. The key advantage of this approach is that the deployment algorithm is entirely distributed, and has the potential to respond dynamically to changes in the environment. This algorithm does, however, lead to relatively slow deployment; it is also unclear, from the published results, how effective this algorithm is at producing good area coverage. A somewhat similar algorithm based on artificial potential fields is described in (Howard et al., 2002c).

The deployment problem described here is similar to that described by (Bulusu et al., 2001), 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, 2000) considers the problem of distributed sensing in an ad-hoc wireless network. Nodes are introduced into the environment en masse 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 line-of-sight network connectivity, and 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 (Balch and Hybinette, 2000; Fredslund and Matarić, 2001; Scheider et al., 2000), 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 in this paper 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 (O'Rourke, 1987). 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 them 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 determine the structure of the environment empirically and incrementally.

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. The algorithm relies on a number of key assumptions, as follows.

Homogeneous nodes: all nodes are assumed to be identical. We also 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. Note that the deployment process *itself* will modify the environment, since deployed 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; indeed, a key task for the network may be to *generate* such models.

Full communication: we assume that all nodes in the network can communicate with some remote base-station on which the deployment algorithm is executed. Note that this does not automatically imply that *all* nodes must be within radio range of the base-station: the nodes may, for example, form an ad-hoc multi-hop network (Intanagonwiwat et al., 2000).

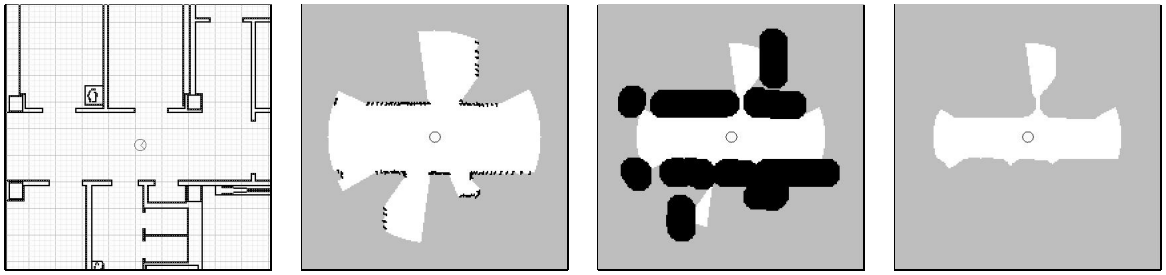
Localization: we assume that the pose of each node is known in some arbitrary global coordinate system. In our previous work on *team localization* (Howard et al., 2002b; Howard et al., 2002d), we have shown how nodes may be localized using only the measured relationships between them. This technique does not require external landmarks or prior models of the environment, but does require that each node is visible to at least one other node. It is this requirement that gives rise to the *visibility constraint*, i.e., each node must be visible to at least one other node at its deployed location. Note that this constraint does not necessarily imply that nodes must be visible *at all times*: we assume that nodes are equipped with some form of odometry or inertial navigation that allows them to localize themselves during periods when they cannot be seen.

We evaluate the incremental deployment algorithm using two performance metrics: *coverage*, i.e., the total area visible to the network's sensors; and *time*, i.e., the total deployment time, including both the time taken to perform the necessary computations and the time taken to physically move the nodes. Naturally, we wish to maximize the coverage while minimizing the deployment time.

3.1. 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*,



(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.

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 a waiting node, and the node's state is changed from *waiting* to *active.* More commonly, assignment is complicated by the fact that deployed nodes tend to obstruct waiting nodes, necessitating a more complex assignment algorithm. That is, the algorithm may have to re-assign the goals of a number of previously 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.

3.2. SELECTION

The selection phase determines the next deployment location, or goal. Ideally, this goal should maximize the coverage metric while 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. The algorithm described here therefore avoids such reasoning altogether, and instead uses a number of relatively simple goal selection *policies*, each of which relies on heuristics to guide the selection process.

As a first step, sensor data from the deployed nodes are combined to form an *occupancy grid* (Elfes, 1987; Elfes, 1990). Each cell in this grid is assigned one of three states: *free*, *occupied* or *unknown*. A cell is *free* if it is known to contain no obstacles, *occupied* if it is known to contain one or more

obstacles, and *unknown* otherwise. We use a standard Bayesian technique (Elfes, 1990) to determine the *probability* that each cell is occupied, then threshold this probability to determine the state of each cell.

In the combined occupancy grid, 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. We can therefore ensure that the visibility constraint is satisfied by always selecting goals that lie somewhere in free space. Not all free space cells represent valid deployment locations, however: since nodes have finite size, they cannot be placed near cells that are either occupied or unknown (unknown cells may 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 free space through an opening that is too narrow to allow passage. To facilitate this kind of analysis, we post-process the occupancy grid to form both a *configuration grid* and a *reachability grid*.

As the name suggests, the configuration grid is a representation of the nodes' *configuration space* (Lozano-Perez and Mason, 1984). 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. 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 resultant 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 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 reachable space, we simultaneously ensure that the deploying node will be visible to at least one other node, that it will not be in collision with the environment, 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 the final goal selection: a *boundary* heuristic and a *coverage* heuristic. Both heuristics operate on the reachability grid. The boundary heuristic states that nodes should deploy to the boundary between reachable and unreachable space; this heuristic effectively minimizes the overlap between adjacent sensory field by placing the nodes as far apart as possible. The coverage heuristic states that nodes should deploy to the (reachable) location from which they will cover the greatest area of presently unknown space (in the occupancy grid). 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, these heuristics do not necessarily specify a unique goal. They can, however, be incorporated into a number of unique goal selection *policies*. We have implemented four such policies:

- **P1:** randomly select a location in reachable space.
- **P2:** randomly select a location on the reachable/unreachable boundary.

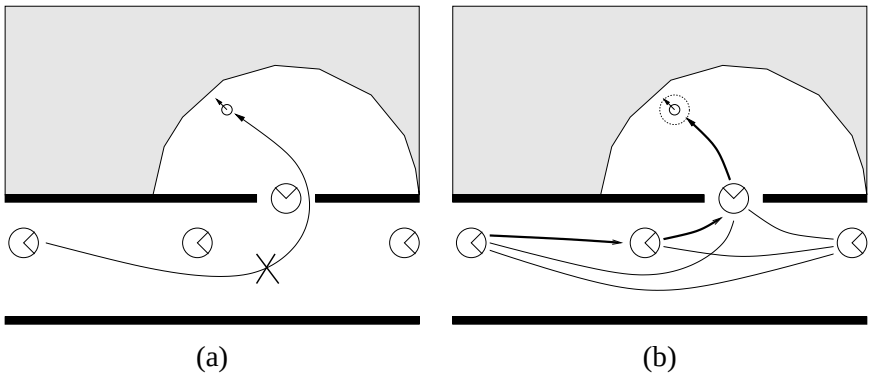


Figure 2. (a) A typical obstruction problem, with a waiting node unable to reach its deployment location. The gray area indicates the region of space that is not yet covered by the network. (b) The obstruction is resolved by re-assigning the deployment location to another node.

– **P3:** select the reachable space location that maximizes the coverage heuristic.

– **P4:** select the reachable/unreachable 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 (policy P1). The first two polices are stochastic, while the latter two are deterministic. Note also 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.

3.3. ASSIGNMENT

The assignment phase of the algorithm 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 by 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 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 that avoids the need 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 vise

versa). 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 a waiting node to the goal. The length of any path through the graph is given by the sum of edge lengths, and the shortest path is found using Dijkstra's algorithm.

– 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, the potential obstruction has been resolved. Note also that while any number of deployed robots may change locations, the sequence of movements alway terminates in the deployment of exactly one waiting robot.

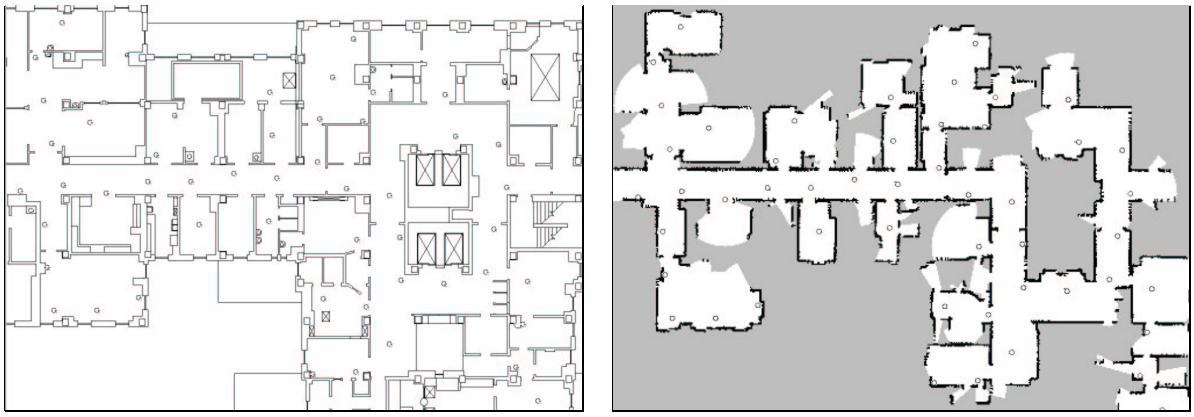
The assignment algorithm requires that we determine the reachability relationship and distance between $n(n-1)/2$ pairs of nodes. In principle, this requires that we generate a *plan* for reaching every node from every other node. In practice, we can simplify this process by generating a unique *distance transform* (Zelinksy, 1992) for each node. The distance transform is generated using a variant of Dijkstra's algorithm (Dijkstra, 1959): distances are propagated out from the goal node, traveling through free configuration space and around occupied or unknown cells. Ultimately, a distance is 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 nodes spread throughout the environment, the same nodes will tend to remain on the edge of the network. It possible, of course, to design different assignment algorithms which generate quite different behavior (such as 'leap-frog' or 'bounding overwatch'); such algorithms are, however, beyond the scope of this paper.

3.4. 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 second 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 assignment 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 area covered by the deployed network becomes larger, nodes will have farther to travel, and hence we expect that execution time will increase with the number of deployed nodes.

Note that 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. With some appropriate modifications to the assignment phase of the



(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).

algorithm, it is possible, in principle, to create an overall 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 (the area covered by the network) and time (both computation and execution). In both cases, we are interested not only in the specific properties of the 50-node network used in these experiments, but also in the implied scaling properties of the algorithm. That is, based on these experiments, 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 (Gerkey et al., 2001) in combination with the Stage multi-agent simulator (Vaughan, 2000). 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 consisted of 50 nodes, each equipped with a 360 degree scanning laser range finder mounted on a differential mobile robot base. Each node was also equipped with an 'ideal' localization sensor that provides accurate position and orientation information. The simulated nodes were placed in the environment shown in Figure 3; this is a fragment of a much larger environment representing 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 points; sensor range was taken to be 2, 4, 6 or 8m. For the stochastic policies P1 and P2, 10 trials were 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

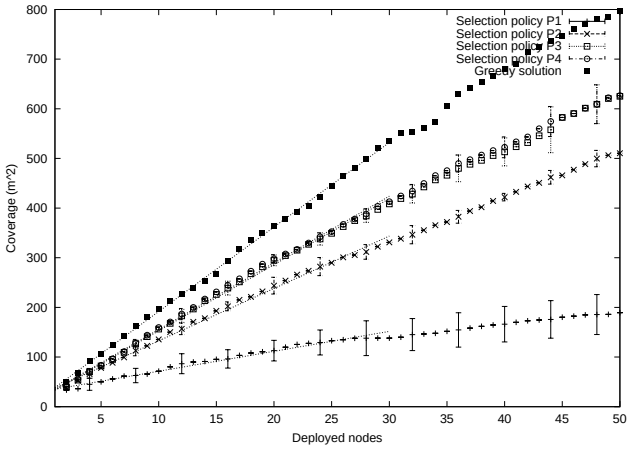


Figure 4. Network coverage for selection policies P1 to P4; the sensor range is 4m. Note that most of the error bars have been suppressed for the sake of clarity.

Table I. Coverage factors for selection policies P1 to P4 with a sensor range of 2, 4, 6 and 8m.

	Range			
Policy	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

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. The 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 3.2, 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 I lists the coverage factors for sixteen different combinations of selection policy and sensor range (determined using 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 was very large and boundary effects have minimal impact (although one can possibly see the start of such effects in some of the coverage plots in Figure 4).

Inspecting the values in Table I, it is apparent that the three goal selection policies that incorporate one or more of the heuristics described in Section 3.2 (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 reachable/unreachable 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: occlusions, not sensor range, will dominate the placement of nodes. Inspecting Table I, 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': this distance may, for example, correspond 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 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 necessitates 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 (O'Rourke, 1987)), we are not aware of any such solutions at this time.

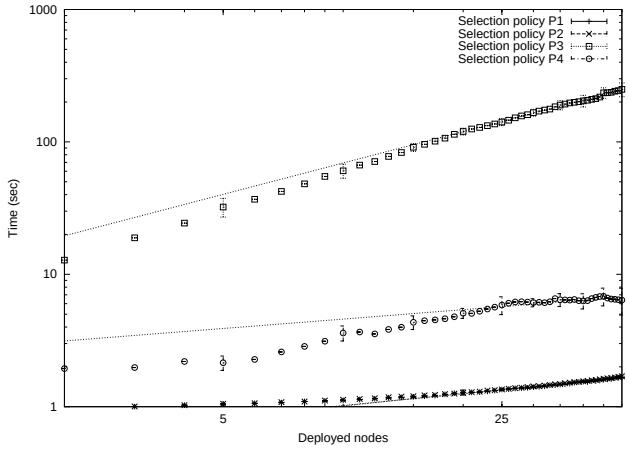


Figure 5. Selection time (CPU) for policies P1 to P4. The scale is log-log. Most of the error bars have been suppressed for the sake of clarity.

Therefore, instead of comparing our results with the optimal solution, we 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. The greedy solution is, in addition, a fairer test for our algorithm: it represents the best result that can be expected for any form of *incremental* deployment algorithm.

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. At the end of this processes, the node is teleported back to the location that produces the greatest coverage, and the process is repeated for the next node.

Table I 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 as good as they are likely to get for a model-free algorithm.

4.2. TIME

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 II 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 (we are trying to capture the highest-order term only).

Table II. 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.

	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

Inspecting this table, two results are immediately apparent. First, and most important, the 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 boundary 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 therefore attribute the sub-linear results in Table II 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 II 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.

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 II lists the a and b values for the assignment phase: it is apparent that this phase scales as n^2 in the number of deployed nodes n. This conforms 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 shown in Section 4.1), the assignment phase will necessarily scale as $n \times n = n^2$.

Note that,we would ideally prefer this phase of the algorithm to scale linearly or better; we are actively seeking alternative assignment algorithms with this property.

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

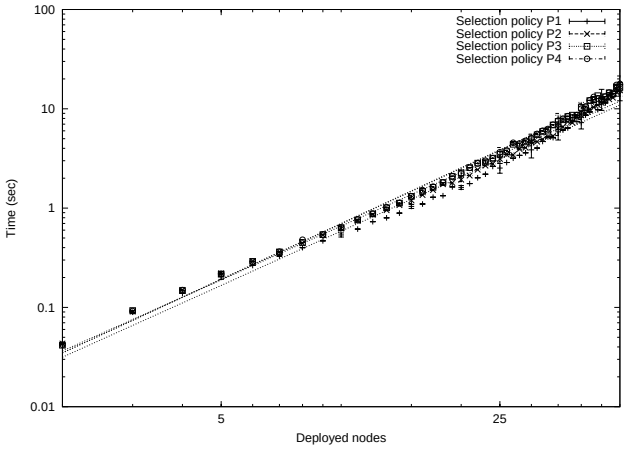


Figure 6. Assignment time (CPU) for policies P1 to P4. The scale is log-log. Most of the error bars have been suppressed for the sake of clarity.

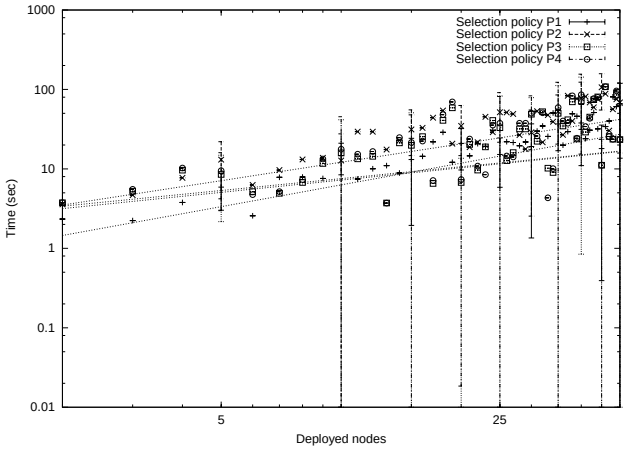


Figure 7. Execution time (wall-clock) for policies P1 to P4. The scale is log-log. Most of the error bars have been suppressed for the sake of clarity.

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 linear, suggesting once again that execution time is a polynomial function of the number of deployed nodes. Table II 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 II 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.

Ideally, we would 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-clock time. If execution time is linear, this rate will necessarily decrease as the number of deployed nodes grows; network coverage will therefore increase only logarithmically with time. Linear growth in coverage 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 3.4, 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.

4.3. DISCUSSION

The experiments described above clearly establish the utility of the incremental deployment algorithm and the heuristics on which it is based. The coverage values for policies P3 and P4 are between 70% and 85% of the value obtained for a model-based greedy algorithm. 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, of course, their reliance on an idealized localization system, in which the pose of each node is accurately determined. With more realistic localization, we expect the algorithm's performance to degrade: poor estimates for the pose of individual nodes will result in poor registration of range data in the combined occupancy grid. This will, in turn, result in poor selections for deployment locations and/or assignment sequences. We do, however, expect the algorithm to degrade gracefully rather than catastrophically, since it does not rely on precise coordination between nodes. Note also that in separate experiments on team localization (Howard et al., 2002b; Howard et al., 2002d) we have demonstrated the ability to accurately localize the nodes in a mobile sensor network, using only the nodes themselves as landmarks, *even when all of the nodes are in continuous motion*. Thus, we are highly optimistic that these two techniques – deployment and team localization – can be merged into an integrated system that exhibits near-optimal performance.

5. Conclusion and Further Work

The incremental deployment algorithm described in this paper can be used to deploy mobile sensor networks into unknown environments. The algorithm will deploy nodes such that network coverage

is maximized while full line-of-sight connectivity is maintained. Furthermore, the algorithm does not require prior models of the environment yet is able to produce coverage results that are close to those obtained using a greedy model-based algorithm.

In addition to the simulation results described in Section 4, we have also taken the first steps to demonstrating the 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 (Howard et al., 2002a), and we are currently preparing a much more ambitious experiment involving up to 9 nodes. Thus we expect to demonstrate the utility of the incremental deployment algorithm for real applications in real environments.

Acknowledgements

The authors would like to thank the anonymous reviewers for a number of useful insights and suggestions.

This work was supported by the DARPA MARS Program grant DABT63-99-1-0015, ONR grant N000140110354, ONR DURIP grant N00014-00-1-0638 and NSF grant ANI-0082498 (SCOWR).

References

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

Bulusu, N., J. Heidemann, and D. Estrin: 2001, 'Adaptive Beacon Placement'. In: *Proceedings of the Twenty First International Conference on Distributed Computing Systems (ICDCS-21)*. Pheonix, Arizona.

Burgard, W., M. Moors, D. Fox, R. Simmons, and S. Thrun: 2000, 'Collaborative Multi-Robot Exploration'. In: *Proc. of IEEE International Conferenceon Robotics and Automation (ICRA)*, Vol. 1. pp. 476–81.

Dedeoglu, G. and G. S. Sukhatme: 2000, 'Landmark-based Matching Algorithms for Cooperative Mapping by Autonomous Robots'. In: L. E. Parker, G. W. Bekey, and J. Barhen (eds.): *Distributed Autonomous Robotics Systems*, Vol. 4. Springer, pp. 251–260.

Dijkstra, E. W.: 1959, 'A Note on Two Problems in Connection with Graphs'. *Numerische Mathematik* **1**, 269–271.

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

Elfes, A.: 1990, 'Occupancy Grids: A Stochastic Spatial Representation for Active Robot Perception'. In: *Proceedings of the Sixth Conference on Uncertainty in AI*. Morgan Kaufmann Publishers, Inc.

Fredslund, J. and M. J. Matarić: 2001, 'Robot Formations Using Only Local Sensing and Control'. In: *International Symposium on Computational Intelligence in Robotics and Automation (IEEE CIRA 2001)*. Banff, Alberta, Canada.

Gage, D. W.: 1992, 'Command Control for Many-Robot Systems'. In: *AUVS-92, the Nineteenth Annual AUVS Technical Symposium*. Hunstville Alabama, USA, pp. 22–24. Reprinted in Unmanned Systems Magazine, Fall 1992, Volume 10, Number 4, pp 28-34.

Gerkey, B. P., R. T. Vaughan, K. Støy, A. Howard, G. S. Sukhatme, and M. J. Matarić: 2001, 'Most Valuable Player: A Robot Device Server for Distributed Control'. In: *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS01)*. Wailea, Hawaii, pp. 1226–1231.

Howard, A., M. J. Matarić, and G. S. Sukhatme: 2002a, 'An Incremental Deployment Algorithm for Mobile Robot Teams'. In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*. EPFL, Switzerland. To appear.

Howard, A., M. J. Matarić, and G. S. Sukhatme: 2002b, 'Localization for Mobile Robot Teams Using Maximum Likelihood Estimation'. In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*. EPFL, Switzerland. To appear.

Howard, A., M. J. Matarić, and G. S. Sukhatme: 2002c, 'Mobile Sensor Network Deployment using Potential Fields: A Distributed, Scalable Solution to the Area Coverage Problem'. In: *Distributed Autonomous Robotic Systems 5: Proceedings of the 6th International Conference on Distributed Autonomous Robotic Systems (DARS02)*. Fukuoka, Japan, pp. 299–308.

Howard, A., M. J. Matarić, and G. S. Sukhatme: 2002d, 'Team Localization: A Maximum Likelihood Approach'. *IEEE Transactions on Robotics and Autonomous Systems*. Submitted May 2002.

Intanagonwiwat, C., R. Govindan, and D. Estrin: 2000, 'Directed Diffusion: A Scalable and Robust Communication Paradigm for Sensor Networks'. In: *Proceedings of the Sixth Annual International Conference on Mobile Computing and Networks (MobiCOM 2000)*. Boston, Massachusetts, pp. 56–67.

López-Sánchez, M., F. Esteva, R. L. de Mántaras, C. Sierra, and J. Amat: 1998, 'Map Generation by Cooperative Low-Cost Robots in Structured Unknown Environments'. *Autonomous Robots* **5**(1), 53–61.

Lozano-Perez, T. and M. Mason: 1984, 'Automatic synthesis of fine-motion strategies for robots'. *International Journal of Robotics Research* **3**(1), 3–24.

O'Rourke, J.: 1987, *Art Gallery Theorems and Algorithms*. New York: Oxford University Press.

Payton, D., M. Daily, R. Estkowski, M. Howard, and C. Lee: 2001, 'Pheromone Robotics'. *Autonomous Robots* **11**(3), 319–324.

Rekleitis, I. M., G. Dudek, and E. E. Milios: 2000, 'Graph-Based Exploration using Multiple Robots'. In: L. E. Parker, G. W. Bekey, and J. Barhen (eds.): *Distributed Autonomous Robotics Systems*, Vol. 4. Springer, pp. 241–250.

Scheider, F. E., D. Wildermuth, and H.-L. Wolf: 2000, 'Motion coordination in formations of multiple mobile robots using a potential field approach'. In: L. E. Parker, G. W. Bekey, and J. Barhen (eds.): *Distributed Autonomous Robotics Systems*, Vol. 4. Springer, pp. 305–314.

Simmons, R., D. Apfelbaum, W. Burgard, D. Fox, M. Moors, S. Thrun, and H. Younes: 2000, 'Coordination for Multi-Robot Exploration and Mapping'. In: *Proc. of the Seventeenth National Conference on Artificial Intelligence (AAAI-2000)*. pp. 852–858.

Thrun, S., W. Burgard, and D. Fox: 2000, '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)*, Vol. 1. pp. 321–328.

Thrun, S., D. Fox, W. Burgard, and F. Dellaert: 2001, 'Robust Monte Carlo Localization for Mobile Robots'. *Artificial Intelligence Journal* **128**(1–2), 99–141.

Vaughan, R. T.: 2000, 'Stage: a multiple robot simulator'. Technical Report IRIS-00-393, Institute for Robotics and Intelligent Systems, University of Southern California.

Winfield, A. F.: 2000, '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 (eds.): *Distributed Autonomous Robotics Systems*, Vol. 4. Springer, pp. 273–282.

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

Yamauchi, B., A. Shultz, and W. Adams: 1998, 'Mobile Robot Exploration and Map-Building with Continuous Localization'. In: *Proceedings of the 1998 IEEE/RSJ International Conference on Robotics and Automation*, Vol. 4. San Francisco, U.S.A., pp. 3175–3720.

Zelinksy, A.: 1992, 'A Mobile Robot Exploration Algorithm'. *IEEE Transactions on Robotics and Automation* **8**(2), 707–717.

Andrew Howard is a research associate in the Computer Science Department at the University of Southern California (USC). He received his Ph.D. in Engineering from the University of Melbourne in 1999, and his B.Sc. (Hons.) in Theoretical Physics from the University of Melbourne in 1991. He was the recipent of an Australian Postgraduate Research Award (Priority), and winner of the best student paper award at the first Field and Service Robotics conference in 1997 (FSR'97). His research interests include multi-robot localization and coordination, distributed sensor/actuator networks, and large-scale simulation of robotic systems.

Maja J Matarić is an associate professor in the Computer Science Department and the Neuroscience Program at the University of Southern California, Director of the USC Robotics Research Lab and Interaction Lab, and Associate Director of IRIS (Institute for Robotics and Intelligent Systems). She received her PhD in Computer Science and Artificial Intelligence from MIT in 1994, her MS in Computer Science from MIT in 1990, and her BS in Computer Science from the University of Kansas in 1987. She is a recipient of the NSF Career Award, the IEEE Robotics and Automation Society Award Early Career Award, the MIT TR100 Innovation Award, and the USC School of Engineering Junior Research Award, and is one of the scientists featured in the film "Me and Isaac Newton." She is an editor of the IEEE Transactions on Robotics and Automation, the International Journal of Autonomous Agents and Multi-Agent Systems, and Adaptive Behavior, has published over 30 journal articles, 7 book chapters, 65 conference papers, and 20 workshop papers, and has two books in the works with MIT Press. She has been involved with NASA's Jet Propulsion Lab, the Free University of Brussels AI Lab, LEGO Cambridge Research Labs, GTE Research Labs, the Swedish Institute of Computer Science, and ATR Human Information Processing Labs. Her research is in the areas of control and learning in behavior-based multi-robot systems and skill learning by imitation based on sensory-motor primitives.

Gaurav S Sukhatme is an Assistant Professor in the Computer Science Department at the University of Southern California (USC) and the associate director of the Robotics Research Laboratory at USC. He received his Ph.D. in Computer Science from USC in 1997. His research interests include embedded systems, mobile robot coordination, sensor fusion for robot fault tolerance, and human-robot interfaces. He directs the Robotic Embedded Systems Lab, which performs research in two related

areas: 1) The control and coordination of large numbers of distributed embedded systems, and 2) The control of systems with complex dynamics (hopping robots, robotic helicopters and haptic interfaces). Dr. Sukhatme is a member of AAAI, IEEE and the ACM and has served on several conference program committees. He has published over 50 technical papers in the areas of robot control, mobile robot cooperation and coordination, and distributed embedded systems.