Cover Me! A Self-Deployment Algorithm for Mobile Sensor Networks

Andrew Howard and Maja J Matarić

Abstract— **This paper describes an algorithm for deploying a mobile sensor network. A mobile sensor network is made up of a distributed collection of nodes, each of which has sensing, computation, communication and locomotion capabilities. In this paper, we describe an *incremental* deployment algorithm in which nodes are deployed one-at-a-time into an unknown environment. Each node makes use of information gathered by previously deployed nodes to determine its optimal deployment location. The algorithm is designed to maximize network 'coverage' whilst 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* [9], [10] we have shown how nodes can localize themselves in a completely unknown environment by using other nodes as landmarks). In this paper, we describe a series of experiments (conducted in both simulation and reality) aimed at validating the algorithm and illuminating its empirical properties.**

I. 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 facilities 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 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 is transmitted to a base station located safely outside the building, where it is assembled to form a live map showing the concentration of hazardous compounds within the building.

For a sensor network is 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 [9], [10], it is possible to determine the location of nodes in a network by using the nodes themselves as landmarks. This technique does, of course, 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 will show in Section III-C, 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 a series of experiments in simulation aimed at characterizing the performance of the incremental deployment algorithm. We have also conducted limited experiments with a real sensor network to confirm the efficacy of the algorithm under controlled real-world conditions.

II. 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 system was introduced by Gage [6]. Gage defines three basic types of coverage: blanket coverage, where the object is to achieve a static arrangement of nodes that maximizes the total detection area; barrier coverage, where the object 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.

Andrew Howard and Maja J Matarić are with the Robotics Research Lab in the Computer Science Department at University of Southern California, E-mail: {ahoward@usc.edu, mataric@usc.edu}

The problem of exploration and map-building by a single robot in an unknown environment has been considered by a number of authors [19], [20], [21]. The frontier-based approach of Yamauchi et al. [19], [20] 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 [3], [15], [16], [12] who use a variety of techniques ranging from topological matching [3] to fuzzy inference [12] and particle filters [16]. Once again, there are two key differences between this earlier work 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).

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 [18] 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.

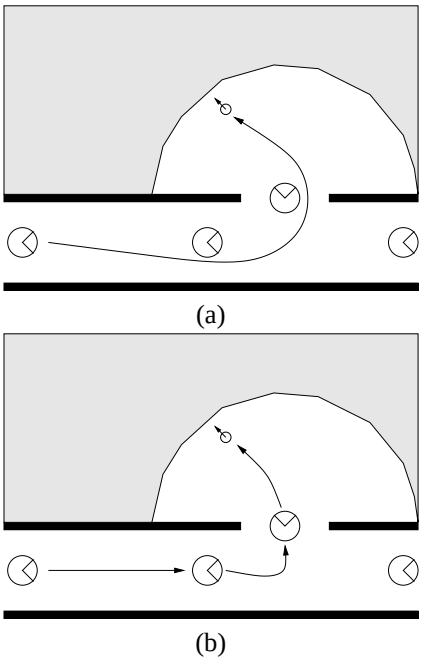


Fig. 1. (a) A typical obstruction problem, with a waiting node unable to reach its deployment location. The gray area indicates the region of space that not yet covered by the network. (b) The obstruction is resolved by re-assigning the deployment location to another node.

Finally, we note the problem of deployment is related to the traditional *art gallery* problem in computational geometry [14]. 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.

III. 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.

A. Assumptions, Constraints, Performance

The incremental deployment 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* [9], [10], we showed 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 constraint:** each node must be visible to at least one other node at its final deployed location.

This constraint does not necessarily imply that nodes must be visible *at all times*: if we assume that the nodes are equipped with some form of odometry or inertial navigation, they need not be continuously visible whilst they are in motion.

The incremental deployment algorithm is designed to maximize a single performance metric:

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

Ideally, we would like to compare the networks produced by the incremental deployment algorithm with an *optimal* network. By definition, an optimal network maximizes the coverage metric whilst simultaneously satisfying the visibility constraint. We would like to be able to indicate, for example, that the coverage for a particular network is 50% that of the optimal network. Unfortunately, finding the optimal network is extremely difficult, even in the presence of a good a priori model. The space of possible networks is vast: consider a network of n nodes in an environment of area a. If we discretize this area into da distinct locations, the number of possible networks is $(da)^n$ (not all of which will satisfy the visibility constraint) For a relatively small network with $n = 10$, $a = 100m^2$ and $d = 10$, the number of possible networks is 10^30. A brute force search is therefore clearly impractical. While there may exist closed form solutions or good approximations for this problem (it is, for example, similar to the art gallery problem [14]), we are not aware of any such solutions at this time. Consequently, in this paper, we make no attempt to find optimal networks.

There is one final issue that must be considered in the design of a practical deployment algorithm: obstruction. When the size of the nodes is comparable with the size of openings in the environment, nodes may find themselves unable to reach their deployment locations due to obstruction from previously deployed nodes. See, for example, the situation illustrated in Figure 1. There is, fortunately, a very natural solution to this problem that exploits the homogeneity of the network nodes: an obstructed node may swap roles with the node obstructing it. Thus, if node A is obstructed by node B, node B will move to A's deployment location, while A will replace B at its original deployment location (see Figure 1). Since all nodes are assumed to be equivalent, this role-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. We will describe a recursive resolution algorithm based on this concept in Section III-D.

B. Algorithm Overview

The incremental deployment algorithm has four phases: initialization, goal selection, goal resolution 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.
- **Goal selection.** Sensor data from the deployed nodes is combined to form a map of the environment. This map is analyzed to select the optimal deployment location for the next node. Ideally, the goal location will maximize the coverage metric whilst simultaneously satisfying the visibility constraint.
- **Goal resolution.** The goal is assigned to a node using a variant of the recursive resolution strategy described previously. If the goal is assigned to a deployed node, the resolution algorithm is called recursively to re-assign the deployed node's previous goal. If the goal is assigned to a waiting node, the algorithm terminates. The state of any node that is assigned or re-assigned a goal is changed to active. Thus it is possible for more than one node to be marked as active during the resolution phase.
- **Execution.** The 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, resolution and execution phases, terminating only when all nodes have been deployed.

C. Goal Selection

The selection phase determines the next deployment location, or goal. Ideally, the goal should maximize the coverage metric whilst simultaneously satisfying the visibility constraint. Unfortunately, in practice, 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 of the environment, we must instead rely on sensor data from deployed nodes; since this data is, by definition, incomplete, our reasoning must be similarly incomplete. Consequently, the algorithm described here avoids such reasoning altogether. Instead, we use one of 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 is combined to form an *occupancy grid* [4], [5]. 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 [5] 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 cells may also turn out to be occupied.

To simplify this kind of analysis, we post-process the occupancy grid to form a *configuration grid*. As the name suggests, the configuration grid is a representation of a node's *configuration space* [13]. As with the occupancy grid, 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*.

[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.

Any point in free space in the occupancy grid is guaranteed to be visible; in contrast, any point in free space in the configuration grid is guaranteed to be both *visible* and *reachable*. Consequently, when selecting a goal, the selection algorithm always chooses a location that lies in free space in the configuration grid.

The selection algorithm makes use of two heuristics to guide 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. By placing the nodes on the *boundary* of free space (in the configuration grid), this heuristic also ensures that the visibility constraint is satisfied.

- **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, if 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.

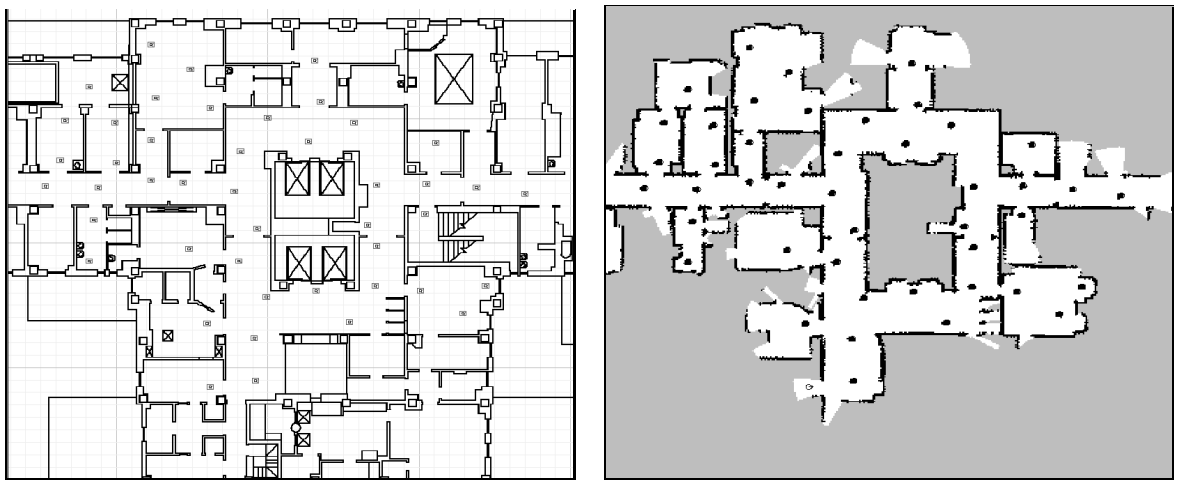
In and of themselves, the heuristics do not necessarily specify a unique goal location. They can, however, be incorporated into a number of goal selection *policies*, each of which will determine a unique goal location. We have implemented four such selection policies:

- **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 IV, we will compare the performance of these four policies in an experimental context, and attempt to determine the relative contributions of the underlying heuristics.

D. Goal Resolution

The goal resolution phase attempts to assign the newly selected goal to a waiting node. As we noted in previously, this process is complicated by the fact that deployed



(a) (b)

Fig. 2. (a) A fragment of the simulated environment. (b) Occupancy grid produced by a typical deployment.

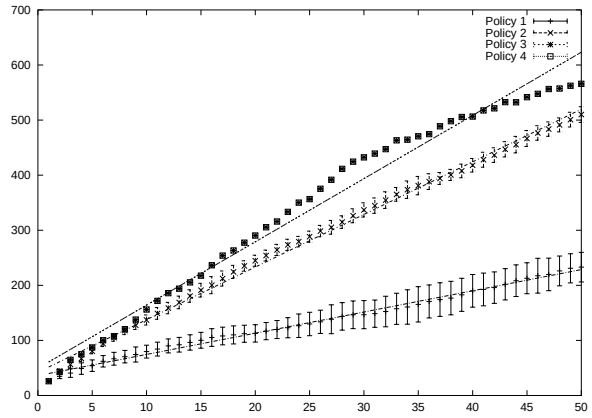


Fig. 3. Coverage versus number of deployed nodes for the simulated environment. The results for all four selection policies are shown, together with the line of best fit for each policy. Note that policies P3 and P4 are not distinguishable in this plot.

nodes tend to obstruct waiting nodes, preventing them from reaching the goal. Thus, it is generally not sufficient to simply assign the goal to the first waiting node. Instead, we use the following *recursive resolution algorithm*.

- Find the nearest deployed or waiting node that can reach the goal.
- If the node is waiting, assign the goal to the selected node, and change the node's state to active.
- If the node is deployed, assign the goal to the selected node, change its state to active, and call the resolution algorithm recursively to assign the node's current location to another node.

This algorithm always assigns the goal to the nearest node, whether it is waiting or deployed (active nodes are not considered, since they already have goals). This produces some interesting behavior: the network will tend to 'ooze' out from its starting location, with many nodes in motion 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.

This 'oozing' behavior can be radically altered through a slight modification to the resolution algorithm. If we bias the algorithm such that goals are assigned to waiting nodes in preference to deployed nodes, the network will tend to deploy in a 'leap-frog' fashion, with most of the nodes remaining stationary while a single node deploys. With this modification, nodes on the edge of the network are continually replaced. This is only true, of course, for relatively open environments, where obstructions are rare. In a highly constrained environment, where obstructions are common, we expect that this modified algorithm will, of necessity, also produce 'oozing' behavior.

It is unclear, at this time, which of these two resolution algorithms is preferable. While the first algorithm may permit faster deployment (the average distance each node must move will be significantly shorter), the second may facilitate better localization (errors tend to accumulate while nodes are in motion, hence keeping most of the nodes stationary should lead to better localization). In this paper, for no particular reason, we will persist with the first of these algorithms.

Note that both resolution algorithms require that we generate, for each node, a *plan* for reaching the goal. To this end, we make use of the configuration grid generated in the selection phase, applying a *distance transform* [21] to the grid, and thereby determining, for each cell, the distance to the goal. The distance transform works by assigning a distance of 0 to the goal cell, a distance of 1 to cells adjacent to the goal, a distance of 2 to cells adjacent to these cells, and so on. Distances are not propagated through occupied or unknown cells. In this way, the distance transform will

TABLE I

COVERAGE RESULTS FOR THE FOUR GOAL SELECTION POLICIES; α MEASURES THE AVERAGE AREA COVERED BY EACH NODE.

Policy	α (m^2)	$1/\alpha$ (m^{-2})
P1	3.85	0.260
P2	9.55	0.105
P3	11.48	0.087
P4	11.48	0.087

assign a value to every cell from which the goal can be reached. Thus, for each node, we can determine the distance to the goal, and whether or not the node can reach the goal, by inspecting the value assigned to the cell containing the node.

E. Execution

During the execution phase, active nodes are deployed to their goal locations. By default, this deployment is *sequential*: we wait for each node to reach its goal location before deploying the next node. Since there is only one node in motion at any given point in time, we eliminate any potential for interference between nodes. Sequential deployment is, however, quite slow. In practice, we find that nodes can be deployed *concurrently* with little or no interference, greatly reducing the overall deployment time. Unfortunately, since we cannot *guarantee* that nodes will not interfere, sequential deployment is always the safer option.

Nodes navigate using a standard potential field controller [1], [11] in which the goal is represented as an attractive force and obstacles are represented as repulsive forces. In order to avoid local minima, we use the gradient of the distance transform generated in the resolution phase to represent the goal force. Thus, the node will tend to 'surf' down the distance transform gradient to the goal. The obstacle force is generated directly from the node's sensors, and prevents the node from colliding with obstacles that where not present when the distance transform was generated. This ensures safe navigation, even when the environment is not entirely static. Such is the case when nodes are deployed concurrently rather than sequentially; the obstacle force can therefore be viewed as another mechanism for reducing interference between nodes.

IV. EXPERIMENTS

We have conducted a series of experiments in simulation and a single experiment with real hardware. The simulation experiments are intended to check the functional validity of the incremental deployment algorithm, and allow us to perform a statistical comparison of the four goal selection policies described in Section III-C. The real-world experiment is intended mainly as a proof-of-concept, establishing that the algorithm is robust in the presence of real sensor and actuator noise.

A. Simulation experiment

The simulation experiments were conducted using the Stage multi-agent simulator [17], [7]. 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 this experiment consists of 50 nodes, each of which is equipped with a scanning laser range finder mounted on a differential mobile robot base. The laser range finder has a 360 degree field-of-view and a maximum range of 4m. 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 [9], [10], as this technique has not yet been merged with the incremental deployment algorithm. The simulated nodes were placed in the environment shown in Figure 2. This is a fragment of a much larger environment that represents a single floor in a large hospital. The initial node placement corresponds to the location of the elevator shaft in the real environment.

We conducted 20 deployment trials for each of the the goal selection policies described in Section III-C. In each trial, we measured the network coverage as a function of the number of deployed nodes. The results are summarized in Figure 3, which shows the coverage (averaged across all trials) plotted against the number of deployed nodes. The plots also show the variance in coverage. Note that the variance for policies P3 and P4 is effectively zero, as one would expect given that these are deterministic policies. In contrast, the variance for policies P1 and P2, which are stochastic, is significant.

Consider the four curves show in Figure 3. Since all four curves are approximately linear, we can determine, for each policy, a value α that measures the average area 'covered' by each node; i.e. α is such that the total network coverage is approximately equal to αn, where n is the number of deployed nodes. The α values for policies P1 through P4 are shown in Table I. It should be noted that these measures are only meaningful 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.

Comparing these results, it is clear that the three goal selection policies that incorporate one or more of the heuristics described in Section III-C (policies P2 to P4) perform significantly better than the control case (policy P1). Policies P3 and P4, in fact, produce a 3-fold improvement over simple random deployment. It is also apparent that most of this improvement can be achieved using the boundary

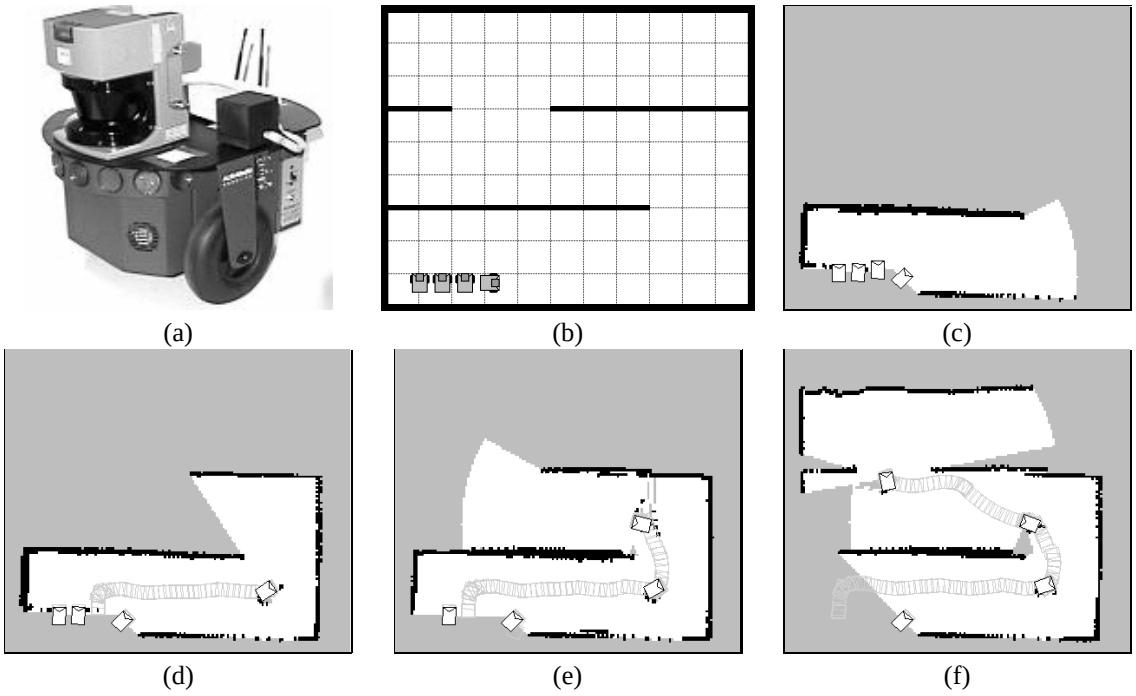


Fig. 4. (a) A sensor node: a SICK scanning laser range-finder mounted on a Pioneer 2DX mobile robot. (b) Initial experimental configuration. (c, d, e, f) The occupancy grid with one, two, three and four deployed nodes.

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 policy P3, since the latter requires much more time to compute and produces negligible improvement in network coverage.

These results also suggest that there may still be room for some improvement in the deployment algorithm. The upper limit on α for a laser range-finder with a $360°$ field-of-view and range of 4m is $50.27\mathrm{m}^2$; our best policies are achieving around one-fifth of this value. While we do not expect this upper bound to be achievable in practice (nor in principle, for that matter, since this bound ignores packing considerations) we would like to explore the relationship between α, sensor range and environmental complexity. This topic is, unfortunately, beyond the scope of this paper.

B. Real-world experiment

We have conducted a single real-world experiment intended to establish that the deployment algorithm will function with real hardware. Our network consists of four nodes, each of which is made up of a SICK LMS200 scanning laser range-finder (with a $180°$ field of view) mounted on a Pioneer 2DX mobile robot. The nodes have an on-board Pentium-class processor and communicate using 802.11 wireless Ethernet. Each node runs the Player [8], [7] robot server; Player abstracts the node's physical hardware into a series of 'devices', which it makes available as network services. Thus, the nodes can be controlled remotely over the network. For this experiment, all four nodes were controlled by a single 450MHz PIII workstation running Linux.

Each of the nodes is also equipped with a beacon-based localization 'sensor'. This virtual sensor processes data from the laser range finder, searching for coded beacons (our beacons are simple bar-codes made from alternating strips of retro-reflective and non-retro-reflective paper). For this experiment, four such beacons were placed in the environment, allowing the nodes to determine their position and orientation to an accuracy of about 10cm and $5°$. As was the case in the simulation experiments, this localization technique is used in place of the mesh-based localization method described in [9], [10].

The environment for this experiment was an artificial one, constructed in the laboratory from wooden partitions. The layout of the environment is shown in Figure 4. Since this environment is less than 7m across, we artificially limited the range of the laser range finders to 4m rather than their usual 8m, in order the make the deployment more difficult.

We conducted a single deployment trial using policy P3. Figure 4 shows a series of 'snap-shots' taken during the trial. Each snap-shot shows the occupancy grid gen-

erated by the deployment algorithm, with the position of each node superimposed. The path taken by nodes between snap-shots is also shown. The trial starts with four nodes in the bottom-left corner of the environment, with the right-most node being used to anchor the network (i.e. this nodes remains stationary). Nodes deploy sequentially, pushing back the free/unknown with each successive deployment. Note that since the topology of the environment is effectively linear, the nodes end up moving in a 'Conga line': as the lead node moves forward, the node immediately behind it steps forward to take its place; this node is in turn replaced by the one behind it, and so on.

While this experiment is limited in scope, it clearly demonstrates that the incremental deployment algorithm can be implemented on real hardware and function under controlled real-world conditions.

V. Conclusion and Further Work

The experiments described in Section IV clearly establish the utility of the incremental deployment algorithm and the heuristics on which it is based. Furthermore, while we have not yet fully characterized the scaling properties of the algorithm, we have empirically demonstrated that this is a practical algorithm for networks containing up to 50 nodes (our simulation experiments were performed in real-time on a single workstation).

The key weakness of these experiments is their reliance on global localization mechanisms 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 [9], [10], this method is not yet integrated with the incremental deployment algorithm described here. We are currently performing this integration, and expect to demonstrate a combined system in the very near future.

Our experiments are also far from exhaustive. There remain many, many issues to explore, including: how does the algorithm scale with network size (in terms of computational cost, bandwidth requirements, and physical deployment time)? How does the algorithm perform in different environments? And what is the impact of changing the sensor range or the physical size of the nodes (thereby reducing obstruction)? All these issues remain the subject of on-going research.

Our work to date demonstrates that a relatively simple deployment strategy, which requires no a priori knowledge of the environment, has the desired properties of robustness and effective coverage; this an important combination for the domains in which these networks will be practically applied.

References

[1] R. C. Arkin. Motor schema based mobile robot navigation. *International Journal of Robotics Research*, 8(4):92–112, 1989.

[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] 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.

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

[5] 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.

[6] 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.

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

[8] 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.

[9] 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.

[10] 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.

[11] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *International Journal of Robotics Research*, 5(1):90–98, 1986.

[12] 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.

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

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

[15] 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.

[16] 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.

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

[18] 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.

[19] 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.

[20] 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.

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