To appear in *Telecommunication Systems Journal, Special Issue on Wireless Sensor Networks,* 26(2-4), 2004

Coverage, Exploration and Deployment by a Mobile Robot and Communication Network

Maxim A. Batalin and Gaurav S. Sukhatme

Robotic Embedded Systems Lab
Computer Science Department
Center for Robotics and Embedded Systems
University of Southern California
Los Angeles, CA 90089

Abstract. We consider the problem of coverage and exploration of an unknown dynamic environment using a mobile robot. The environment is assumed to be large enough such that constant motion by the robot is needed to cover the environment. We present an efficient minimalist algorithm which assumes that global information is not available (neither a map, nor GPS). Our algorithm deploys a network of radio beacons which assists the robot in coverage. The network is also used by the robot for navigation. The deployed network can also be used for applications other than coverage (such as multi-robot task allocation). Simulation experiments are presented which show the collaboration between the deployed network and mobile robot for the tasks of coverage/exploration, network deployment and maintenance (repair), and mobile robot recovery (homing behavior). We discuss a theoretical basis for our algorithm on graphs and show the results of the simulated scenario experiments.

1 Introduction

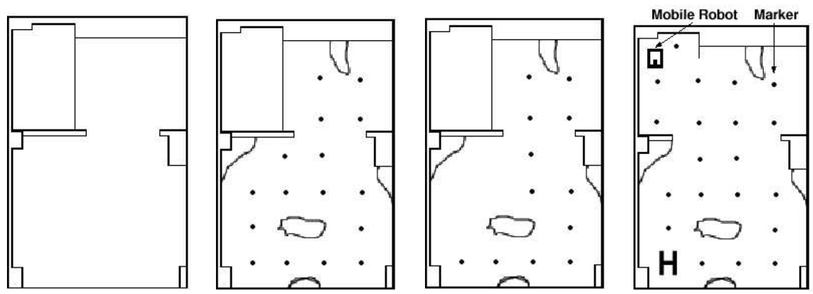
We consider two problems from traditionally different backgrounds. The first is the **exploration and coverage of a space by a mobile robot**. The coverage problem has been defined [1] as the maximization of the total area covered by the robot's sensors. There are many applications of coverage such as tracking unfriendly targets (e.g military operations), demining or monitoring (e.g. security), and urban search and rescue (USAR) in the aftermath of a natural or man-made disaster (e.g. building rubble due to an earthquake or other causes). We require the robot to cover all areas of the space, and to occasionally navigate to a designated target location in the space. The second problem is the **deployment of a sensor and communication network** into an environment. Such a network may be used for monitoring, or as an ad-hoc communication infrastructure. Our claim is that these two problems are best solved together i.e. a *combined* solution exists which satisfies both objectives. The basic idea is simple - the robot deploys the network into the environment as it explores, and the network guides future robot exploration.

Coverage can be considered as a *static* or a *dynamic* problem. The *static* coverage problem is addressed by algorithms [2–4]. The goal of these algorithms is to converge to a static configuration (an equilibrium state), such that every point in the environment is under the robots' sensor shadow (i.e. covered) at every instant of time. For complete static coverage of an environment the robot team should have a certain critical number of robots (depending on environment size, complexity, and robot sensor ranges). Determining the critical number is difficult or impossible [2] if the environment is unknown *a priori*. *Dynamic* coverage, on the other hand, is addressed by algorithms which explore and hence 'cover' the environment with constant motion and neither settle to a particular configuration [5], nor necessarily to a particular pattern of traversal. Coverage of the environment can be accomplished over time with any number of robots.

In this paper we consider the case of a single robot in an environment that is large enough that complete *static* coverage of the environment is not possible. The robot must thus continually move in order to observe all points in the environment frequently. In other words, we study the *dynamic* coverage problem with a single robot. We briefly discuss various multi-robot extensions at the end of the paper.

Single robot exploration of unknown environments has been studied before [6–8]. The frontier-based approach [6, 7] incrementally constructs a global occupancy map of the environment. The map is analyzed to locate the 'frontiers' between the free and unknown space. Exploration proceeds in the direction of the closest 'frontier'. The multi-robot version of the same problem was addressed in [9]. The problem of coverage was considered from the graph theoretic viewpoint in [10, 11]. In both cases the authors study the problem of *dynamic* single robot coverage on an environment consisting of nodes and edges (a graph). The key result was that the ability to tag a limited number of nodes (in some cases only one node) with unique *markers* dramatically improved the cover time. It may be noted that both papers consider the coverage problem, but in the process also create topological maps of the environment graph being explored. Using markers for robot navigation has also been the subject of research in biologically-inspired robotics where the markers form a trail [12] (inspired by the trail-laying behavior of ants).

The algorithm we propose (a variation of the *Node Counting* and *Edge Counting* algorithms discussed in [13, 14]) differs from the above approaches in a number of ways. We use neither a map, nor localization in a shared frame of reference. Our algorithm is based on the deployment of a set of static nodes into the environment by the robot. The nodes form a communication network. We term every node in the network *a marker*. The markers we use act as a support infrastructure, which the mobile robot uses to solve the coverage problem efficiently. The robot explores the environment, and based on certain *local criteria*, drops a marker into the environment, from time to time. Each marker is equipped with a small processor and a radio of limited range. Our algorithm performs the coverage task successfully using only local sensing and local interactions between the robot and markers. The approach builds on our prior work [5], and strives to maintain connectivity in the network.



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

Fig. 1. A schematic of a) Initial Environment (before the experiment); b) Environment after changes with deployed network(beginning of experiment); c) Some of the nodes require replacement (malfunctioned, damaged, etc.); d) Another alteration to environment and a robot that has to return to marker H;

Our key contribution is an algorithm for robot exploration and coverage that relies on the deployment of a communication network. Once deployed the network is used by the robot for efficient exploration and navigation. We note that our approach for navigation is similar to [15], which uses potential fields whereas we use value iteration.

Analysis of the deployed network as a graph shows that our algorithm is *complete* i.e. it covers every vertex of the graph and *efficient* (cover time linear in the size of the network graph). We discuss data from one long term continuous experiment which includes a dynamic environment and exhibits the major functionalities of our approach: the ability to provide full coverage/exploration of the environment, robustness to changes in the environment, ability to replace damaged markers, navigation and extensions to multi-robot applications utilizing the deployed network.

2 Experimental Scenario

Imagine a scenario where the environment changes dramatically in a short timespan; for example a collapsing building. In such a situation a mobile robot, or a group of robots, could be sent into the building to search for people. Our system allows a mobile robot to explore (and completely cover) the environment without access to a prior map, by deploying markers into the environment. Subsequently, the robot is able to 'home' to a given location using the same set of markers.

Figure 1a shows the floor plan of the environment prior to changes. Conventional approaches to covering this environment and exploring it, could use a map-based technique (such as the ones in [6, 7]). Suppose however that due to a

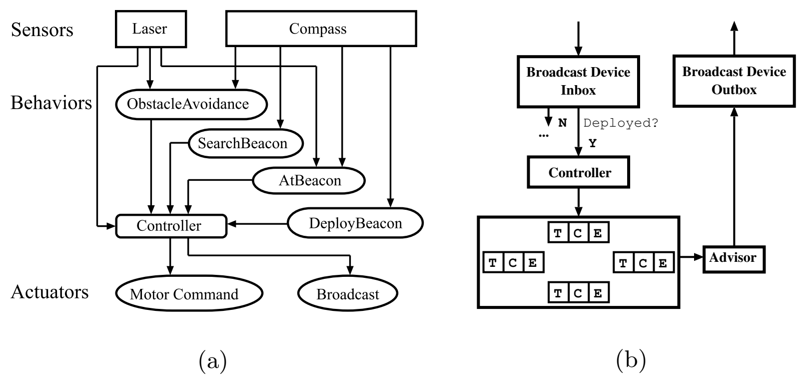


Fig. 2. a) System Architecture showing Robot Behaviors; b) Beacon Architecture

catastrophic event (e.g. earthquake, fire) debris is introduced into the environment, thereby altering it (Figure 1b). Even though the map of the environment might be available initially, an altered environment would be difficult or impossible to cover and explore, with approaches relying on metric/topological map usage. The experimental work reported in this paper starts at this point. A robot is introduced into the environment of Figure 1b. The robot explores the environment by populating it with markers that form a network. Figure 1c shows a schematic of the network with some of the nodes removed (malfunctioned, destroyed, etc.). Using our algorithm, the robot repairs the gap in the network by deploying new nodes. The last step of the scenario is depicted in Figure 1d. The environment was altered again so that extra space in the environment is uncovered. The robot is now required to explore and cover the extra space by deploying markers. In addition, the robot is required to use deployed network for homing - returning to a special marker (H on Figure 1d).

3 Architecture

Our algorithm uses two entities: the markers and the mobile robot. The task of each marker is to recommend a locally preferred direction of movement for the robot within its communication range. Thus each marker acts as a local signpost telling the robot which direction to explore next. The robot treats this information as a recommendation, and combines it with local range sensing (to avoid obstacles) to make a decision about which direction to actually pursue.

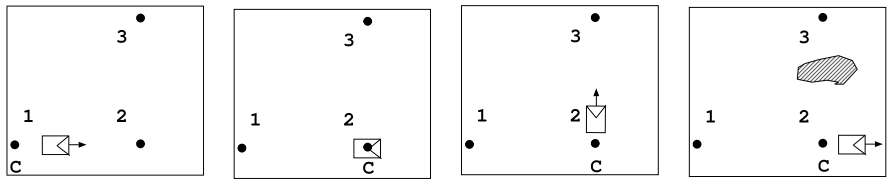
As shown in Figure 2(b), each marker has a state associated with four cardinal directions (South, East, North, West). The choice of four directions is arbitrary. It implies that each marker is equipped with a 2 bit compass. For each direction, the marker maintains a binary state (T), a counter (C) and block E which might be used for additional information. The state T can be either `OPEN` or

`EXPLORED`, signifying whether the particular direction was explored by the robot previously. The counter C associated with each direction stores the time since that particular direction was last explored.

When deployed, a marker emits two data packets with different signal strengths. The packet with the lower signal strength is called the *MIN*-packet and the one with the higher signal strength is called the *MAX*-packet. The *MAX*-packet is used for data propagation within the deployed network. We discuss it in section 5.2. The *MIN*-packet contains two pieces of information: a) the suggested direction the robot should take for coverage/exploration and b) the suggested direction the robot should take for homing. This implies that the robot's compass and the marker's compass agree locally on their measurement of direction. Given the coarse coding of direction we have chosen, this is not a problem in realistic settings. The algorithm used by the markers to compute the suggested direction for exploration/coverage is a 'least recently visited direction' policy. All `OPEN` directions are recommended first (in order from South to West), followed by the `EXPLORED` directions with largest last update value (largest value of C). Note that this algorithm does not use inter-marker communication. The computation of the suggested direction for homing is discussed in a later section (section 5.1).

The robot uses a behavior-based approach [16] with arbitration [17] for behavior coordination. Priorities are assigned to every behavior *a priori*. As shown in Figure 2(a), the robot executes four behaviors: *ObstacleAvoidance*, *AtBeacon*, *DeployBeacon* and *SearchBeacon*. In addition to priority, every behavior has an activation level, which decides, given the sensory input, whether the behavior should be in an active or passive state (1 or 0 respectively). Each behavior computes the product of its activation level and corresponding priority and sends the result to the Controller, which picks the maximum value, and assigns the corresponding behavior to command the Motor Controller for the next command cycle.

During motion, the robot maintains the notion of a current marker (Figure 3a). This is the node whose *MIN*-packets are received by the robot most frequently. When the robot moves to the vicinity of a new marker, the *AtBeacon* behavior is triggered and the robot's current marker is updated (Figure 3b). *AtBeacon* analyzes the *MIN*-packets received from the current marker and orients the robot along the suggested direction contained in those packets. In addition, the robot sends an update message to the marker telling it to mark the direction from which the robot approached it as `EXPLORED`. This ensures that the direction of recent approach will not be recommended soon. We term this the *last-neighbor-update*. After the robot has been oriented in a new direction, it checks its range sensor for obstacles. If the scan does not return any obstacles, the robot proceeds in the suggested direction (Figure 3c), while sending a message to its current marker updating the state of the suggested direction to `EXPLORED` (the marker also resets the corresponding C value). If, however, the suggested direction is obstructed, the *AtBeacon* behavior updates the marker with this information and requests a new suggested direction (Figure 3d). The



(a) Current Beacon is 1

(b) Current Beacon is 2

(c) Choice of direction (no obstacles)

(d) Choice of direction (with obstacles)

Fig. 3. Behavior Switching. a) The robot is executing *SearchBeacon* behavior traversing suggested direction; b) The robot is executing *AtBeacon* behavior, analyzing sensor readings; c) The robot is executing *SearchBeacon* behavior, supposing the beacon suggests direction *UP* and there are no obstacles detected in the sensor data; d) The robot is executing *SearchBeacon* behavior traversing in direction, not originally suggested by the marker.

Obstacle Avoidance behavior is triggered if an obstacle is detected in front of the robot, in which case an avoidance maneuver takes place.

Once the robot is oriented in a new direction (whether as a result of taking the advice of the marker, or as a result of avoiding an obstacle), the *SearchBeacon* behavior is triggered. *SearchBeacon* causes the robot to travel a predetermined distance without a change in heading (assuming there are no obstacles in the way). The *DeployBeacon* behavior is triggered if the robot does not receive a *MIN*-packet from any marker after a certain timeout value. In this case the robot deploys a new marker into the environment.

During its exploration of the environment, the robot builds a transition graph. We call this **deployed network graph**. The vertices of the graph represent the deployed markers. A directed edge from vertex A to B is labelled with the probability of arriving at node B from node A by proceeding in a particular direction. In section 5 we discuss the use of this graph for computing probabilistic paths through the environment between any two nodes, and thus, using the marker network for probabilistic navigation.

4 Graph Model

For purposes of analysis, consider an open environment (no obstacles). Given our marker deployment strategy described in the previous section, we can model the steady state spatial configuration of the markers as a regular square lattice. In fact, the analysis applies to any graph of degree 4 isomorphic to a regular lattice. Without loss of generality we ignore the boundary of the graph in the analysis.

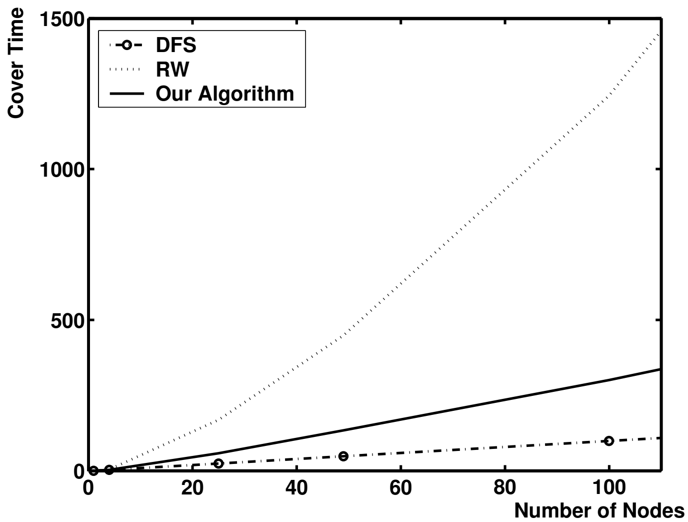


Fig. 4. A comparison between DFS, RW and our algorithm.

In the general case the deployed network graph would be a regular graph of degree 4. The *cover time* [18], is the time it takes a robot to cover (visit) every node in the graph and can be computed as the number of actions taken by the robot to visit every node of the graph. The problem of coverage on the graph is to minimize the average *cover time*, considering every vertex of the graph as a starting point.

We assert that our algorithm covers the environment completely i.e. the robot visits every node of the graph. In the most simple case where the environment is unknown, and localization cannot be used, and there are no *markers* available, the problem of coverage can be solved by a random walk (RW). It has been shown [18] that the *cover time* of a random walk on a regular graph of n nodes is bounded below by $n \ln n$ and above by $2n^2$. If we assume that passive markers can be used, and the graph $G = (V, E)$ is known (a topological map is available) and the robot has *markers* of three independent colors, then the problem of coverage can be solved optimally by applying depth first search (DFS) which is linear in n. DFS assumes that all resources are available - markers, map, localization and perfect navigation.

We conducted experiments running RW, DFS and our algorithm on graphs with $n = 25, 49$ and 100 nodes. For every experiment each grid point was tried as the starting point. We conducted 50 experiments per starting point, such that as soon as robot covers all nodes, the nodes become uncovered and the coverage task starts from the node where the robot finished its last coverage. Then the next starting point is considered and so on. The average cover time over all experiments was computed. The results of this experiment are shown in

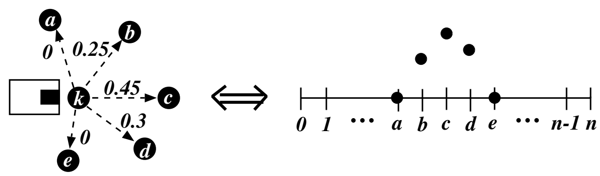


Fig. 5. An example of a discrete probability distribution of vertex (marker) k for direction (action) "East"(i.e. right).

Figure 4; our algorithm and DFS both perform asymptotically better than the RW.

Note that in order to determine the color of neighboring vertices and navigate from one vertex to another, DFS assumes that a topological map of the environment is available and the robot is localized. Our algorithm, on the other hand, does not have access to global information and the robot does not localize itself. The *markers* used in our algorithm are more complicated than those used in DFS, and the cover times are asymptotically somewhat larger than the cover times of DFS.

5 Connectivity Map and Probabilistic Navigation

In order for the robot to be able to navigate through the environment from point A to point B, assuming neither map nor GPS are available, the robot should be able to recognize that it has arrived at the goal (B), be able to measure progress and be able to choose an action that maximizes its chances of getting to its goal.

5.1 Value Iteration

We assume finite set of vertices S in the deployed network graph and a finite set of actions A the robot can take at each node (marker). Given a subset of actions $A(s) \subseteq A$, for every two vertices in the deployed network graph $s, s' \in S$ and $a \in A(s)$ the robot should determine the transitional probability $P(s'|s,a)$ (probability of arriving at vertex s' given that the robot started at vertex s and commanded an action a). In our algorithm four actions are possible at every vertex (marker) - East, West, South and North. Thus, for every action a_i at a given vertex $s \in S$ and all other vertices $s' \in S - s$ the robot computes the probability $P(s'|s,a_i)$ as the ratio of the number of transitions from s to s' with action a_i to the number of times a_i was commanded at vertex s. This ratio is normalized to ensure that $\sum_{a_i} P(s'|s,a_i) = 1$. Figure 5 shows a typical discrete probability distribution for a vertex (marker) per action (direction). Note that in practice the probability mass is distributed around neighboring nodes and zero otherwise.

Our model for the proposed system is Markovian - the state the robot transitions to depends only on the current state and action. We model the navigation problem as a Markov Decision Process [19]. To compute the best action at a given vertex we use the Value Iteration [20] algorithm on the set of vertices $S - s_g$, where s_g is the goal state. The general idea behind Value Iteration is to compute the utilities for every state and then pick the actions that yield a path towards the goal with maximum expected utility. The utility is incrementally computed:

$$U_{t+1}(s) = C(s,a) + \max_{a \in A(s)} \sum_{s' \in S-s} P(s'|s,a) \times U_t(s') \tag{1}$$

where $C(s,a)$ is the cost associated with moving to the next vertex. Usually the cost is chosen to be a negative number which is smaller than $-1/k$ where k is the number of vertices. The rationale is that the robot should 'pay' for taking an action (otherwise any path that the robot might take would have the same utility), however, the cost should not be to big (otherwise the robot might prefer to stay at the same state). Initially the utility of the goal state is set to 1 and of the other states to 0. Given the utilities, an *action policy* is computed for every state s as follows:

$$\pi(s) = \arg\max_{a \in A(s)} \sum_{s' \in S-s} P(s'|s,a) \times U(s') \tag{2}$$

The robot maintains a probabilistic transition model for the deployed network graph, and can compute the action policy at each node for any destination point. In practice however, this is limiting, since it requires the robot to traverse the network many times over to learn the transition model. Further, another robot deployed into the same environment would need to first traverse the deployed network before it can navigate between any two points optimally.

One solution is for the robot to compute the action policy as above, and while traversing the network record the optimal action for the current marker as it passes by. Each marker can store this action and can emit it as part of the direction suggestion packet (see Section 3). This would help other robots (which may not yet have explored the entire space) use the information for navigation. However, this solution is inefficient, since it is slow to adapt if the navigation goal is changed.

5.2 Distributed computation and In-network Processing

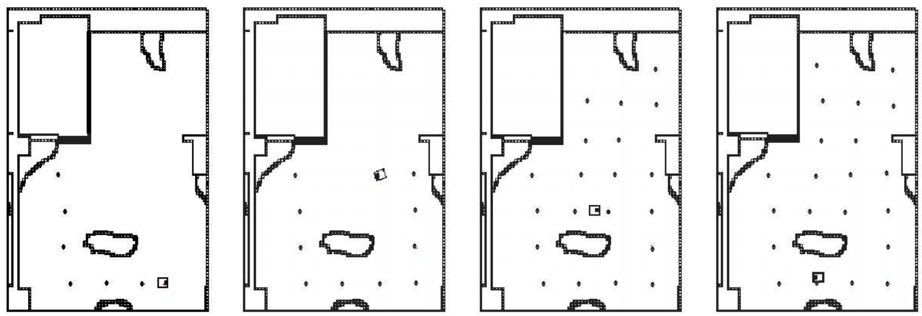
A much more attractive solution is to compute the action policy distributively in the deployed network. The idea is that every node in the network updates its utility and computes the optimal navigation action (for a robot in its vicinity) on its own. While traversing the deployed network the robot stores the transition probabilities $P(s'|s,a)$ on the corresponding markers. Then, if a robot wants to navigate to a point in the environment it injects a *Start Computation* packet into the network containing the target marker's id. Every marker redirects this packet to its neighbors using flooding. Markers that receive the *Start Computation* packet initialize utilities and the cost values depending on whether this particular marker specified as a target or not. Every marker updates the utilities according to equation 1. Note that the utilities of neighboring markers are needed as well, hence, the marker queries its neighbors for corresponding utilities. Since computation of some markers can proceed faster than of the others, every marker stores computed utilities in a list, so that even if it's being queried by its neighbors for a utility several steps prior to the current one, the list is accessed and the corresponding utility is sent.

After the utilities are computed, every marker computes an optimal policy for itself according to equation 2. Neighboring markers are queried once again for the final utility values. The computed optimal action is stored at each marker and is emitted as part of the MIN-packet (refer to section 3) for homing to the goal.

This technique allows the robot to navigate through the environment between any two nodes of the deployed network. Note that the action policy computation is done only once, and does not need to be recomputed, unless the goal changes. Also, note that utility update equations have to be executed until the desired accuracy is achieved. For practical reasons the accuracy in our algorithm is set to 10^{-3}, which requires a reasonable number of executions of the utility update equation per state and thus, the list of utilities that every marker needs to store is small. Since the computation and memory requirements are small it is possible to implement this approach on the real marker device that we are using (the Mote [21]).

6 Simulation Experiment

We conducted a continuous experiment that tests the algorithm for reliability and robustness to environmental changes, problems in the network and shows the ability to deploy and maintain a network and use it for coverage/exploration and navigation. Thus, the scenario consists of four phases. In Phase 1 the robot's task is to deploy a network and cover/explore the environment completely. In Phase 2 we assume that certain nodes in the network failed and require replacement, thus, the goal of the algorithm is to find the gap in the network and replace the damaged nodes, while covering the environment. Phase 3 distorts the environment further, by introducing an extra space - a "hidden room" which also has to be covered. Then, the robot computes the transition probabilities and stores the appropriate constants at every marker. In Phase 4, we assume that another robot appears on the scene, which does not have any prior knowledge about environment and the deployed sensor network. It executes the same algorithm as the robot-deployer, but in this case the part of data packet containing action policy for homing is preferred and used as a suggested direction of the marker. Note that even though the algorithm is robust against loss of some data packets or imprecise compass readings, in simulations we assume that the compass and radio properties are ideal.



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

Fig. 6. Sequential deployment of network.

6.1 First Phase

As shown in Figure 1b, the environment has been altered so that an initial map of the environment would not be useful in coverage. Assuming that a mobile robot with a set of markers have been introduced into the environment (thrown in, dropped by an air vehicle, etc.). The robot starts deployment and coverage/exploration process at the same time. While deploying markers, the robot updates its connectivity map. The deployment of the sensor network for this stage of the scenario is presented on Figure 6 in sequence.

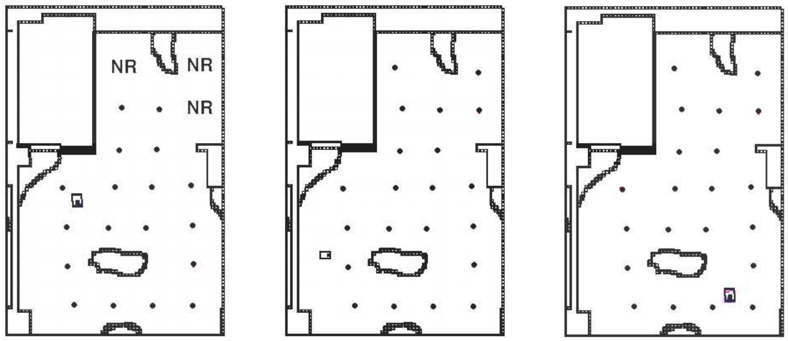
As shown on the above figure, the robot deployed the network over the whole environment, while at the same time accomplishing coverage. Figure 9 represents coverage values over the first three phases of the experiment.

6.2 Second Phase

As shown in Figure 7a, several nodes of the sensor network were removed (nodes in the upper part of the figure are assumed to be malfunctioned or damaged). As seen in the Figure 7, the gap in the network has been detected by the robot and repaired. Note that the robot continued coverage of the environment(Figure 9) and was not affected by the problems in the network.

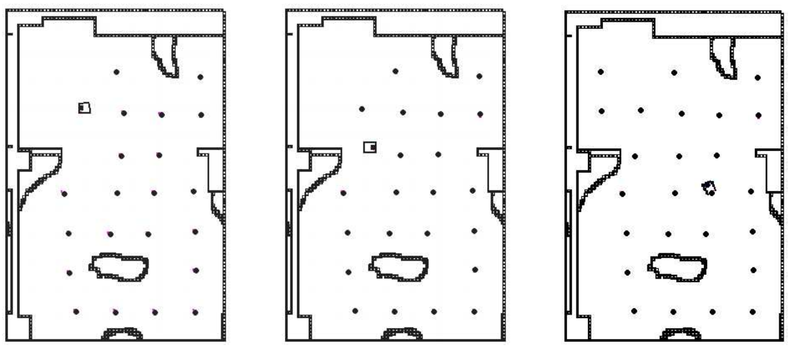
6.3 Third Phase

In this phase of the experiment, we assume that certain perturbations occurred in the environment so that the robot starts with the environment shown in Figure 8a. Figure 8bc show expansion of the network by deployment of additional markers into new open space by the robot. Note, that the problem of coverage was not abandoned by the robot under the circumstances depicted in last three



(a) (b) (c)

Fig. 7. Network repair. NR - area requiring repair



(a) (b) (c)

Fig. 8. Deployment of additional markers into the discovered open space.

phases. A unified view of cover time for three phases is shown in Figure 9. In addition, the robot injects a *Start Computation* packet and the *navigation field* is computed.

6.4 Fourth Phase

In the fourth, last phase, the trapped robot discovered a deployed sensor network. The task is to use the navigational constant and to drive to the home area marked with H (Figure 10a). Figure 10b shows the navigational field that was produced

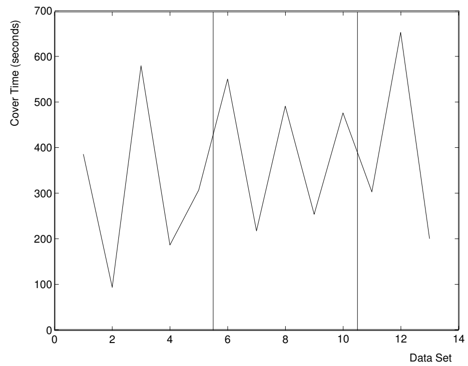
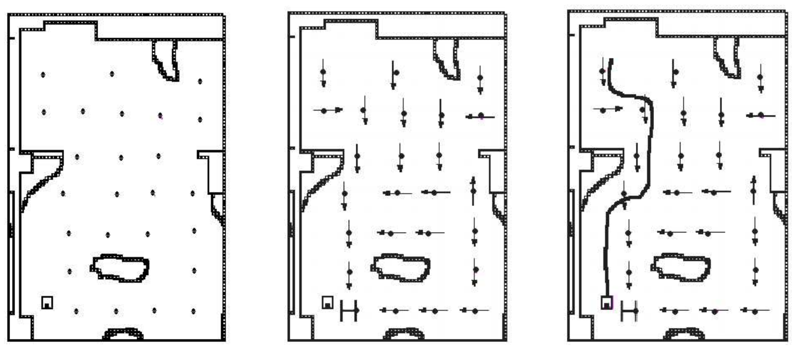


Fig. 9. Coverage over the three stages of the experiment.



(a) Before Navigation

(b) Navigation Field

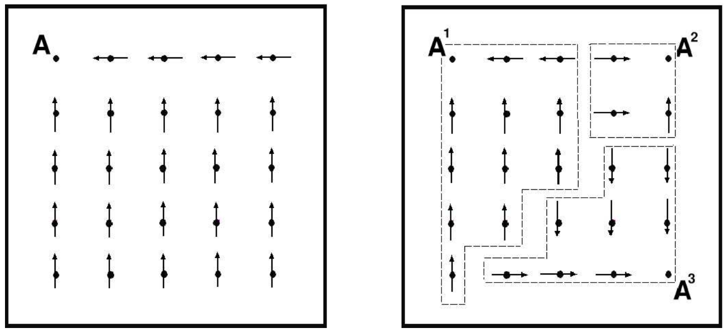
(c) One path taken by the robot

Fig. 10. Robot navigation through the environment

by distributive computation of the optimal policy by the deployed network. The path that the robot traverses is shown on Figure 10c.

7 Multi-robot extensions

Thus far we have discussed the applicability of our system to coverage, deployment, exploration and navigation. However, the deployed sensor network has the potential to be used as a task-manager in the multi-robot domain. The ability of the deployed network to respond to queries by different robots (executing different tasks) allows it to serve as a multi purpose infrastructure. This could



(a) One goal

(b) Three goals; A^1, A^2 and A^3

Fig. 11. Examples of Navigation Field computed by DINTA.

enable, for example, solutions to problems requiring heterogeneous groups of robots. Imagine a scenario on a construction site which requires cooperation of two groups of robots - transporters and builders. Transporters concentrate on delivering the materials to several piles while builders choose the type of material they need at the moment from a corresponding pile and continue construction. Thus, a transporter robot might use the network to find the shortest path towards the material storage or towards the pile that requires certain material the most. While the builder robot would be directed towards a pile with required material or towards another builder needing assistance. In other words, the network can be used as a distributed multi-functional manager, which can also be used for task coordination.

The above application is illustrative of the general *online task allocation problem*. Suppose there are several different tasks that the robots should carry out (transport or build, for example), moreover these tasks arrive in real time. The problem is to allocate resources (robots) to tasks in an efficient manner. In recent work [22] we proposed an approach to solve this problem: Distributed In-Network Task Allocation (DINTA). The core idea of the approach is that given n goals (tasks) in the environment with different importance (i.e. weight, priority, etc.), every node decides which goal it should compute direction to based on its distance to the goal (measured in hop counts) and the importance that the goal has. As a result, the network computes distributively the navigational field, comprised of n different subfields as shown on Figure 11. Robots then are implicitly assigned to the goals based on the particular subfield they are located at. For the multi-robot case, the performance of DINTA [22] is an improvement over the exploration approach discussed in this paper.

We are presently working on an extension of DINTA, where every node computes direction (task assignment) to every goal in the environment and the network explicitly assign each goal to a specific robot. Early simulation results show that as the number of robots increase in the system, this variant of DINTA outperforms the original algorithm, and does not waste resources by assigning different robots to the same goal.

8 Conclusions and Future Work

We presented an algorithm for robot coverage, exploration, and navigation through the utilization of a deployed network. Several capabilities of the algorithm were demonstrated - network deployment and repair, probabilistic navigation, coverage and exploration, and robustness to environmental and network changes. An experimental scenario was executed which tested the above mentioned capabilities. Throughout the execution of the scenario cover time was measured. The cover time shows that despite perturbations to the environment and network, the robot was able to maintain coverage. As mentioned in the previous section, the presented approach is extendable to multi-robot applications, in which the network can be thought of as a multi-purpose task manager.

A scheme for probabilistic navigation is also presented, however, not yet extensively tested. In this instance, the network assists the robot in navigation by the fact that the robot is always localized within the sensor network, and therefore there is no need for feature detection or prior knowledge of a map. Note, however, that the probabilistic navigation was not incorporated to assist the coverage task itself. A possible extension is for the robot to navigate from an explored subset of nodes to an unexplored subset, which would essentially reduce the problem of coverage to that of search. The proposed probabilistic navigation scheme is distributed, which improves performance and allows robots that do not have prior information about the deployed network to navigate between any two markers in the environment.

The major motivation for our approach is that a static deployed network can be used in collaboration with mobile robots. This allows us to design a minimalist algorithm for robot navigation which does not require a map of the environment or GPS. In addition, metric localization does not take place. The tradeoff is the assumption that the number of available markers is large and that markers are not a scarce resource, a reasonable assumption nowadays [21].

The results presented in this work were conducted in simulation. Figure 12 shows some of the screen shots of a preliminary experiment using hardware. Experiments are in progress using a Pioneer 2DX mobile robot equipped with 180° laser range finder, compass and wireless ethernet and a set of motes (as markers) equipped with CPU, RAM and radio of adjustable signal strength. Experiments and extensions to the multi-robot case are also in progress.

9 Acknowledgment

This work is supported in part by NSF grants ANI-0082498, IIS-0133947, CCR-0120778, and EIA-0121141.

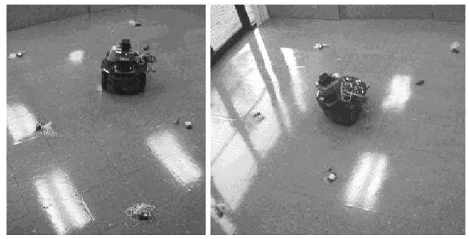


Fig. 12. Screen shots of a preliminary physical experiment.

References

1. Gage, D.W.: Command and control for many-robot systems. In: the Nineteenth Annual AUVS Technical Symposium, Huntsville, Alabama, USA (1992) 22–24
2. O'Rourke, J.: Art Gallery Theorems and Algorithms. Oxford University Press, New York (1987)
3. Howard, A., Mataric, M.J., Sukhatme, G.S.: Mobile sensor network deployment using potential fields: A distributed, scalable solution to the area coverage problem. In: Proc. of 6th International Symposium on Distributed Autonomous Robotic Systems, Fukuoka, Japan (2002) 299–308
4. Batalin, M.A., Sukhatme, G.S.: Spreading out: A local approach to multi-robot coverage. In: Proc. of 6th International Symposium on Distributed Autonomous Robotic Systems, Fukuoka, Japan (2002) 373–382
5. Batalin, M.A., Sukhatme, G.S.: Sensor coverage using mobile robots and stationary nodes. In: SPIE2002. Volume 4868. (2002) 269–276
6. Yamauchi, B.: Frontier-based approach for autonomous exploration. In: In Proceedings of the IEEE International Symposium on Computational Intelligence, Robotics and Automation. (1997) 146–151
7. Yamauchi, B., Schultz, A., Adams, W.: Mobile robot exploration and map-building with continuous localization. In: In Proceedings of the 1998 IEEE/RSJ International Conference on Robotics and Automation. Volume 4. (1998) 3175–3720
8. Zelinsky, A.: A mobile robot exploration algorithm. In: IEEE Transactions on Robotics and Automation. Volume 8. (1992) 707–717
9. Burgard, W., Fox, D., Moors, M., Simmons, R., Thrun, S.: Collaborative multirobot exploration. In: Proc. of IEEE International Conferenceon Robotics and Automation (ICRA). Volume 1. (2000) 476–481
10. Dudek, G., Jenkin, M., Milios, E., Wilkes, D.: Robotic exploration as graph construction. In: IEEE Transactions on Robotics and Automation, 7-6. (1991) 859–865
11. Bender, M.A., Fernandez, A., Ron, D., Sahai, A., Vadhan, S.: The power of a pebble: Exploring and mapping directed graphs. In: Annual ACM Symposium on Theory of Computing (STOC '98). (1998)
12. Vaughan, R., Stoy, K., Sukhatme, G.S., Matarić, M.J.: Lost: Localization-space trails for robot teams. IEEE Transactions on Robotics and Automation, Special Issue on Multi-robot Systems **18** (2002) 796–812
13. Koenig, S., Simmons, R.: Easy and hard testbeds for real-time search algorithms. In: Proccedings of National Conference on Artificial Intelligence. (1996) 279–285

14. Szymanski, B., Koenig, S.: The complexity of node counting on undirected graphs. Technical report, Computer Science Department, Rensselaer Technical Institute, Troy(New York) (1998)
15. Li, Q., DeRosa, M., Rus, D.: Distributed algorithms for guiding navigation across a sensor network. Technical Report TR2002-435, Dartmouth, Computer Science Technical Report (2002)
16. Mataric, M.J.: Behavior-based control: Examples from navigation, learning, and group behavior. Journal of Experimental and Theoretical Artificial Intelligence, special issue on Software Architectures for Physical Agents **9** (1997) 323–336
17. Pirjanian, P.: Behavior coordination mechanisms - state-of-the-art. Technical Report IRIS-99-375, Institute for Robotics and Intelligent Systems, University of Southern California (1999)
18. Lovasz, L. In: Random Walks on Graphs: A Survey. Volume 2 of Combinatorics, Paul Erdos is Eighty., Keszthely, Hungary (1993) 1–46
19. White, D.J.: Markov Decision Process. John Wiley & Sons, West Sussex, England (1993)
20. Koenig, S., Simmons, R.G.: Complexity analysis of real-time reinforcement learning applied to finding shortest paths in deterministic domains. Technical Report CMU-CS-93-106, Carnegie Mellon University, School of Computer Science, Carnegie Mellon University, Pittsburg, PA 15213 (1992)
21. Pister, K.S.J., Kahn, J.M., Boser, B.E.: Smart dust: Wireless networks of millimeter-scale sensor nodes. Electronics Research Laboratory Research Summary (1999)
22. Batalin, M.A., Sukhatme, G.S.: Sensor network-based multi-robot task allocation. In: To appear in Proc. of IEEE/RSJ Intl. Conf. On Intelligent Robots and Systems (IROS'03), Las Vegas, Nevada (2003)