In *IEEE International Conference on Robotics and Automation*
pp. 636-642, New Orleans, LA, April 26 - May 1, 2004

Mobile Robot Navigation using a Sensor Network

Maxim A. Batalin and Gaurav S. Sukhatme
Robotic Embedded Systems Laboratory
Center for Robotics and Embedded Systems
Computer Science Department
University of Southern California
Los Angeles, CA 90089, USA
Email: maxim@robotics.usc.edu, gaurav@usc.edu

Myron Hattig
Intel Corporation
Open Source Robotics
Hillsboro, Oregon
Email: myron.hattig@intel.com

***Abstract*— We describe an algorithm for robot navigation using a sensor network embedded in the environment. Sensor nodes act as signposts for the robot to follow, thus obviating the need for a map or localization on the part of the robot. Navigation directions are computed within the network (not on the robot) using value iteration. Using small low-power radios, the robot communicates with nodes in the network locally, and makes navigation decisions based on which node it is near. An algorithm based on processing of radio signal strength data was developed so the robot could successfully decide which node neighborhood it belonged to. Extensive experiments with a robot and a sensor network confirm the validity of the approach.**

I. Introduction

Navigation is a fundamental problem in mobile robotics. The *local* navigation problem deals with navigation on the scale of a few meters, where the main problem is obstacle avoidance. A well-known solution to this problem is presented in [1], [2], where an occupancy grid map of the immediate surroundings of the robot is created and used to determine the navigation direction such that the robot is safely guided towards a goal. Since the map is local, and resembles a 'sliding window', mapping of the whole environment does not occur.

The *global* navigation problem deals with navigation on a larger scale in which the robot cannot observe the goal state from its initial position. A number of solutions have been proposed in the literature to address this problem. Most rely either on navigating using a pre-specified map or constructing a map on the fly. Most approaches also rely on some technique of localization. Some work on robot navigation is *landmark-based* relying on topological maps [3], which have a compact representation of the environment and do not depend on geometric accuracy. The downside of such approaches is that they suffer from sensors being noisy and the problem of sensor *antialiasing* (i.e. distinguishing between similar landmarks). Metric approaches to localization based on Kalman filtering [4] provide precision, however the representation itself is unimodal and hence cannot recover from a lost situation (misidentified features or states). Approaches developed in recent years based on 'Markov localization' [5] provide both accuracy and multimodality to represent probabilistic distributions of different kinds, but require significant processing power for update and hence are impractical for large environments. One of the attempts to solve this problem is presented in [6] where a sampling-based technique is used. Rather than storing and updating a complex probability distribution, a number of samples are drawn from it. The other approaches utilize partially observable Markov decision process (POMDP) models to approximate distance information given a topological map, sensor and actuator characteristics [7]. POMDP models for robotic navigation provide reliable performance, but fail in certain environments (e.g symmetric) or suffer from large state spaces (i.e. *state explosion*).

These approaches have different advantages, but also disadvantages or fail cases. Note that all of the above approaches assume that a map of the environment (topological and/or metric) is given a priori. None of the above approaches deal with highly dynamic environments in which topology might change. Our approach, presented here, instruments the environment with a sensor network.

An ant-like trail laying algorithm is presented in [8], where 'virtual' trails are formed by a group of robots. Navigation is accomplished through trail following. The shortcoming of the algorithm is that it is dependent on perfect communication between the members of the group. In addition, the 'virtual' trails are shared between the robots, which means redundant sharing of the state space in the group. Moreover, a common localization space is assumed.

We are broadly interested in the mutually beneficial collaboration between mobile robots and a static sensor network. The underlying principle in interaction between the network and robots is: the network serves as the communication, sensing and computation medium for the robots, whereas the robots provide actuation, which is used among other things for network management and updating the network state. In this work we describe results from such a system which accurately and reliably (100% correct navigation out of 50 experiments totaling over 1km in distance) solves the problem of robot navigation. Some properties of the approach are summarized below:

1) The sensor network is predeployed into the environment using the algorithm given in [9].
2) In addition to deploying the network nodes, the deployment algorithm also computes the distributions of transition probabilities $P(s'|s,a)$ from network node s to s', when the robot executes action a [10].

3) The nodes of the sensor network are synchronized in time (high precision is not required). For an example of a time synchronization algorithm see [11].
4) The robot does not have a pre-decided environment map or access to GPS, IMU or a compass.
5) The environment is not required to be static.
6) The robot does not perform localization or mapping.
7) The robot does not have to be sophisticated - the primary computation is performed distributively in the sensor network, the only sensor required is for obstacle avoidance.

The environment used in the experiments is a regular cubicle-like space, with changing topology, narrow corridors (just over 1m) and full of obstacles (people, packaging boxes, trash cans, etc). Note also that usually cubicles or other places of interest in such environments are marked (with laser bar codes, number codes, etc). Hence, existing markers can be complimented with sensor network nodes that are used in our algorithm.

II. Probabilistic Navigation

In order for the robot to be able to navigate through the environment from point A to point B, assuming neither a map nor GPS are available, the robot should be able to choose an action that maximizes its chances of getting to its goal, to be able to measure progress and recognize that it has arrived at the goal (B). Our approach relies on a predeployed sensor network with determined transition probabilities (see below) [10] and consists of two stages.

A. Stage I - Planning

When the navigation goal is specified (either the robot requests to be guided to a certain place, or a sensor node requires the robot's assistance), the node that is closest to the goal triggers the *navigation field* computation. During this computation every node probabilistically determines the optimal direction in which the robot should move, when in its vicinity. The computed optimal directions of all nodes in conjunction compose the *navigation field*. The *Navigation Field* provides the robot with the 'best possible' direction that has to be taken in order to reach the goal. Note that a 'kidnapped' robot problem is solved by our system implicitly and does not require re-computation (or re-planning).

It may be noted that a parallel approach for the construction of a navigation field has been proposed in the sensor network literature [12]. Instead of value iteration [12] uses potential fields and the hop count to compute the magnitude of the directional vectors.

1) Theoretical Framework - Value Iteration: Consider the deployed sensor network as a graph, where the sensor nodes are vertices. Assume a finite set of vertices S in the deployed network graph and a finite set of actions A the robot can take at each node. Given a subset of actions $A(s) \subseteq A$, for every two vertices $s, s' \in S$ in the deployed network graph, and action $a \in A(s)$ the transition probabilities $P(s'|s,a)$ (probability of arriving at vertex s' given that the robot started at vertex s and commanded an action a) for all vertices are determined [10]. Figure 1 shows a typical discrete probability distribution for a vertex (sensor node) per action (direction). Note that in practice the probability mass is distributed around neighboring nodes and zero otherwise.

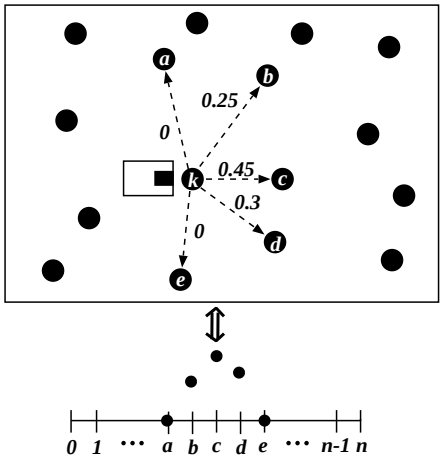


Fig. 1. An example of a discrete probability distribution of vertex (sensor node) k for direction (action) "East"(i.e. right).

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 [13]. To compute the best action at a given vertex we use the Value Iteration [14] 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:

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

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:

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

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 node as it passes by. Each node can store this action and can emit it as part of the message directed to a robot. 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.

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. When the navigation goal is determined (either a robot requested to be guided to a certain node, or a node requires robot's assistance), the node that is closest to the goal triggers the computation by injecting a *Start Computation* packet into the network containing its id. Every node redirects this packet to its neighbors using flooding. Nodes that receive the *Start Computation* packet initialize utilities and the cost values depending on whether the particular node is specified as a goal or not. Every node updates the utilities according to equation 1. Note that the utilities of neighboring nodes are needed as well, hence, the node queries its neighbors for corresponding utilities. Since computation of some nodes can proceed faster than others, every node stores computed utilities in a list, so that even if it is 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 node computes an optimal policy for itself according to equation 2. Neighboring nodes are queried once again for the final utility values. The computed optimal action is stored at each node and is emitted as part of a *suggestion* packet that the robot would receive if in the vicinity of the node.

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 the 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 node needs to store is small. Since the computation and memory requirements are small it is possible to implement this approach on the real node device that we are using (the Mote [15]).

Note that if neighbors of all nodes are known exactly (for every direction each node has at most one neighbor), then $P(s'|s,a) = 1$. Hence, equations 1 and 2 reduce to the maximization of utilities of neighboring nodes only. In this case the system converges after a single iteration.

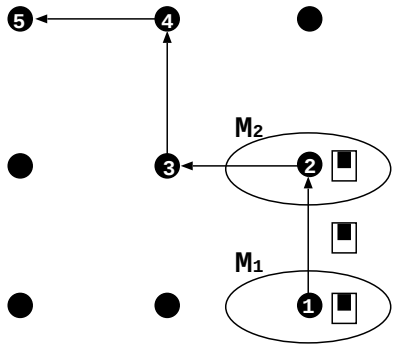


Fig. 2. Navigation - node-wise approach

B. Stage II - Navigation

Note that the deployed sensor network discretizes the environment. Consider Figure 2. On the way from starting node 1 to goal node 5, the robot would first navigate from node 1 to 2, then from 2 to 3, and so on. Hence, the navigation is node-wise. A node whose directional suggestion the robot follows at the moment is called *current node*. Initially the *current node* is set to the node closest to the robot. The bottom part of Figure 2 shows the three phases of navigation. Suppose initially *current node* is set to node 1 (robot's position at the bottom right corner on the Figure 2). Node 1 suggests the robot to go 'UP'. In the first phase the robot accepts this command and positions itself in the correct direction. During the second phase, the robot moves 'forward' using the VFH [2] algorithm for local navigation and obstacle avoidance. Note that throughout the second phase the *current node* is set to node 1. Phase 3 is triggered when the robot determines that it has entered the neighborhood of the next node - say, node 2 (an oval M_2 on Figure 2). During phase 3 the *current node* switches and the navigation algorithm starts from phase 1 again, but with the *current node* set to 2. But how to determine when the robot is in the neighborhood of some node? A straightforward approach is to use signal strength thresholding. In this case, prior to the experiment an observation model can be built which given a signal strength value would approximate the distance from the node. Hence, ideally, while in phase 2, the robot would simply collect signal strength values from the packets of all nodes in the vicinity, feed the model with these values and threshold an output picking the shortest distance.

The problem with such an approach is that raw signal strength values are neither constant nor even proportional from radio to radio, from one environmental topology to another, etc. Experimental results show that such an approach is not reliable or accurate. To reliably predict which node neighborhood the robot is in, we developed an algorithm[1] called Adaptive Delta Percent, based on processing signal strength values in the following manner.

[1]A more general version of the algorithm is submitted for patenting which is based on the idea that any kind of statistical approach can be used to compare the rates of change in signal strength

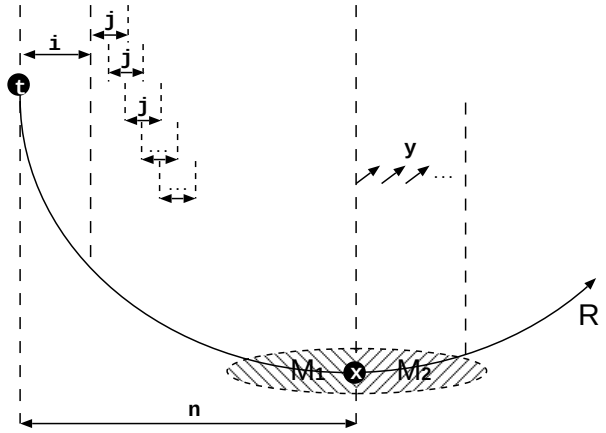


Fig. 3. Illustration of Delta Percent Algorithm. Arc R represents an ideal signal strength space that the robot receives starting from node t, going towards node x. Note that metric data is not involved in the figure and the desired 'switching' place is around areas M_1 or M_2

Consider that the robot's current node is node t. The node suggests to the robot to travel in a certain direction. Assume that before reaching the next node the robot would receive n samples of radio signal strength from each of the k nodes to which the robot might switch to (i.e. *candidate nodes*). Then for each of the k nodes:

1) Compute an *initial maximum average* A_{im} - an average of the first i samples where $i << n$.
2) Compute a *running average* A_r which is an average of j consecutive samples where $j << i$.
3) If $R = \frac{A_r}{A_{im}} < M$, where M is the threshold value, then return from the algorithm. Put R into list L_R.
4) If y consecutive elements of L_R are in nondecreasing order, then return from the algorithm, else repeat 2 through 4.

In case several nodes returned from the algorithm, pick the node with the smallest ratio and switch to it. In our experimental setup $n \approx 15$, $i \approx 5$, $j \approx 10$ and $y \approx 3$. Experimentally we determined threshold $M = 0.65$. Consider Figure 3. An arc R represents ideal signal strength space that the robot would be moving in on its way from node k to some final node x. Note that in reality the signal strength space is not uniform. The desired 'switching' place in the neighborhood of x is around areas M_1 or M_2. Steps 1-3 of the algorithm try to estimate if the robot is in area M_1. Step 4, on the other hand is checking if the robot has passed area M_1 and now is in area M_2. This scenario can happen if threshold M is difficult or incorrectly determined or parameters i and j were chosen inappropriately.

III. Experiments and Discussion

We conducted experiments at Intel Research facilities in Hillsboro, Oregon. We used a Pioneer 2DX mobile robot, with 180^o laser range finder used for obstacle avoidance, and a base station (Mica 2 mote) for communicating with the sensor network. Mica 2 motes were used as nodes of the

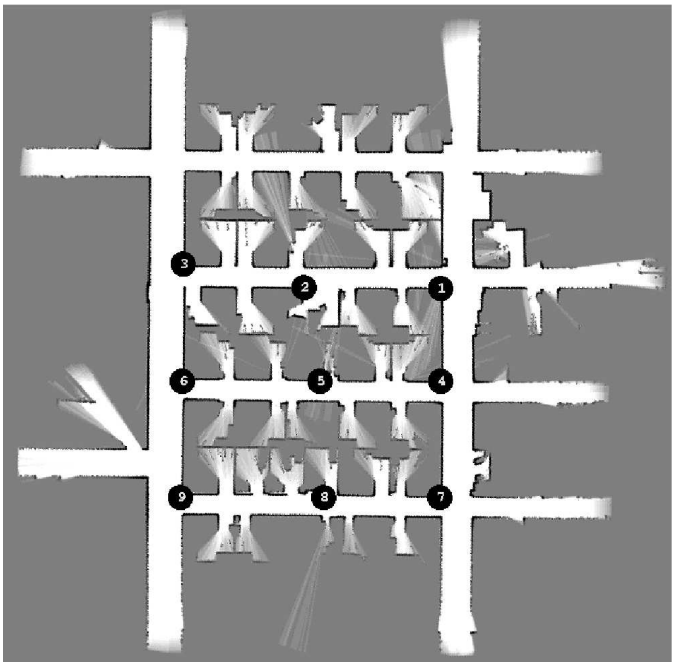
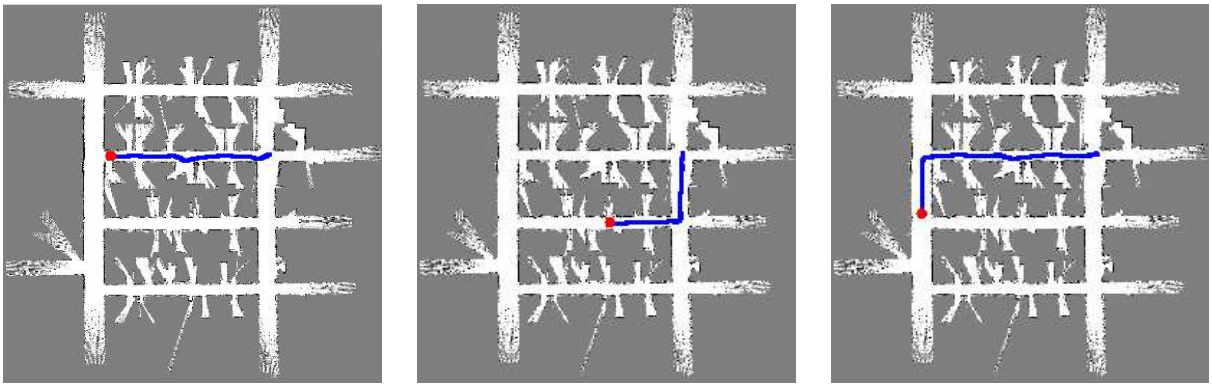


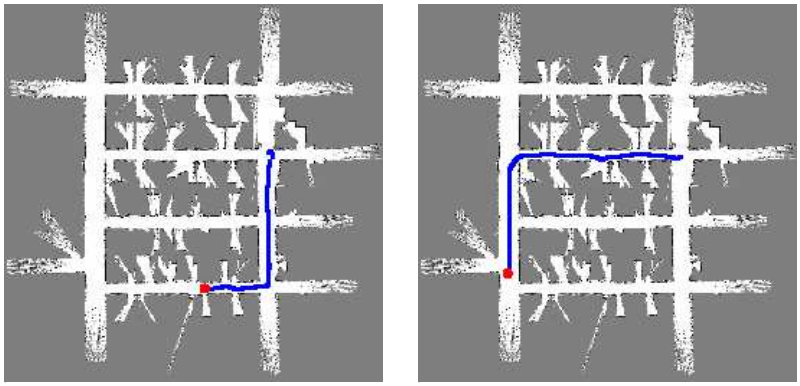
Fig. 4. Map of the experimental environment. Nodes were manually predeployed (nodes marked 1 - 9)

Fig. 5. Mobile robot and a Mote in experimental setting.

sensor network. The sensor network of 9 nodes was predeployed into the environment. Every node is preprogrammed with information about its neighbors. We assume that the sensor network is deployed and transition probabilities set as described in [10]. The map of the experimental environment and deployed sensor network of 9 sensor nodes is shown in Figure 4. The environment itself resembles a regular cubicle-office-like environment with narrow corridors (about 1 m), changing topology, crowded with people and obstacles. Figure 5 shows the mobile robot and one of the deployed nodes in the experimental environment. The experimental scenario that we consider for navigation is *alarm handling*. An *Alarm*



(a) Goal 3 (b) Goal 5 (c) Goal 6



(d) Goal 8 (e) Goal 9

Fig. 6. Trajectories of robot navigating to five different goals. The start location in each case is near node 1.

occurs when a certain node detects an event. The algorithm proceeds as discussed in previous sections. The task of the robot is to navigate from the 'home base' (around node 1) towards the triggered alarm. The requirements that we impose for the experiment to be successful are that the navigation field should yield shortest paths from any point towards the goal node, the robot should follow the shortest path, and the robot should stop within 3 meters of the goal node.

Since the robot does not have an IMU or a compass (in the experimental environment these devices proved to be useless), the direction in which the robot is initially facing is explicitly set. During the experiments the robot maintains the notion of virtual direction, that is, given initially set direction the robot switches virtual direction only when nodes tell robot to switch direction. We conducted 10 experiments for five different goal nodes (we set off an *alarm* at five different nodes) - 3, 5, 6, 8 and 9 (50 experiments altogether). Table I shows the final distances from the robot to the goal nodes after the robot has signaled that it had completed navigation. The length of navigation paths that the robot traveled combined is over 1 km. The robot was able to navigate to the correct goal node in all cases. Representative trajectories that the robot took on its route from the start (node 1) to five goal nodes are depicted in Figure 6.

TABLE I

EXPERIMENTAL DATA (DISTANCE TO GOAL AT FINISH, IN METERS). FIVE GOALS, TEN EXPERIMENTS PER GOAL.

Trial	Goal 3	Goal 5	Goal 6	Goal 8	Goal 9
1	0.7	1.4	0.78	2.9	0.96
2	0.82	1.26	0.86	1.6	0.96
3	0.94	1.45	0.72	1.62	1.35
4	0.91	1.41	0.91	2.4	1.26
5	0.85	1.4	0.87	1.4	1.21
6	0.97	1.39	1.3	2.1	1.24
7	0.85	1.01	0.85	1.7	0.95
8	0.98	1.55	0.88	2.8	1.51
9	0.89	1.5	0.55	1.79	1.4
10	0.66	1.04	1.02	2.1	0.92
Average	0.86	1.34	0.87	2.04	1.17

As the results suggest, our algorithm provides precise and reliable navigation. Neither a map, nor localization was used in the process and GPS, IMU and compass were not available.

The proposed algorithm requires a sensor network to be deployed and moreover, every node of the network should probabilistically know its neighbors. Note also that usually environments resembling our experimental space has every cubicle and intersection marked with either laser bar code or RFID tag. Imagine marking every cubicle or 'interest point' with markers like nodes used in our experiment. In this case the deployment problem is no longer an issue. At the same time, we showed in our earlier work [10] how to deploy a sensor network with a mobile robot and in the process determine transition probabilities for every node.

IV. Conclusion

We have presented an algorithm that allows the robot to navigate precisely and reliably using a deployed sensor network. Our approach differs from systems described in the literature by assuming that a map, localization, GPS, IMU or compass are not available. The navigation occurs through node-wise motion from node to node on the path from starting node to the goal node. We conducted 50 experiments for 5 different goals, totaling over 1 km of traveled distance. In each of the 50 cases the robot successfully navigated to the goal node. Note that we considered an experiment to be successful if the robot approached the goal node to within 3m. This distance was experimentally set as 'good enough', since goal nodes represent a 'local neighborhood' requiring robot's presence. Hence, when the robot arrives at such a 'local neighborhood', local navigation algorithms, like VFH [2], can be used to drive the robot exactly to where the robot's presence is required. Furthermore, in practice, sensor network nodes would be mounted on top of the cubicles (in places where current markers are), which makes the 3m range reasonable.

As can be noted, proposed system contains multiple parameters. In the future work we plan to investigate how accurately can the switching stage be. In addition, we plan to augment the Adaptive Delta Percent algorithm to be able to tune its parameters on-the-fly as the robot proceeds and detects statistical dependence in gathered signal strength values.

Acknowledgment

This work is supported in part by NSF grants ANI-0082498, IIS-0133947, and EIA-0121141 and Intel Corporation. We would like to thank Christopher Jones for implementation of VFH [2] algorithm. We also would like to thank Jim Butler and Edward Epp (Intel Corporation) for the support of the project.

References

[1] I. Ulrich and J. Borenstein, "Vfh+:reliable obstacle avoidance for fast mobile robots," in *IEEE Int. Conf. on Robotics and Automation*, May 1998, pp. 1572–1577.

[2] ——, "Vfh*:local obstacle avoidance with look-ahead verification," in *IEEE Int. Conf. on Robotics and Automation*, April 2000, pp. 2505–2511.

[3] D. Kortenkamp and T.Weymouth, "Topological mapping for obile robots using a combination of sonar and vision sensing," in *Proceedings of the AAAI*, 1994, pp. 979–984.

[4] K. Arras, N. Tomaris, B. Jensen, and R. Siegwart, "Multisensor on-the-fly localization: Precision and reliability for applications," *Robotics and Autonomous Systems*, vol. 34, no. (2-3), pp. 131–143, 2001.

[5] D. Fox, "Markov localization: A probabilistic framework for mobile robot localization and naviagation," Ph.D. dissertation, Institute of Computer Science III, University of Bonn, 1998.

[6] F. Dellaert, D. Fox, W. Burgard, and S. Thrun, "Monte carlo localization for mobile robots," in *Proc. of ICRA-99*, 1999, pp. 1322–1328.

[7] R. Simmons and S. Koenig, "Probabilistic robot navigation in partially observable environments," in *Proceedings of the International Joint Conference on Artificial Intelligence*, 1995, pp. 1080–1087.

[8] R. Vaughan, K. Stoy, G. S. Sukhatme, and M. Mataric, "Lost: Localization-space trails for robot teams," *IEEE Transactions on Robotics and Automation*, vol. 18, no. 5, pp. 796–812, 2002.

[9] M. A. Batalin and G. S. Sukhatme, "Efficient exploration without localization," in *In Proc. of IEEE International Conference on Robotics and Automation (ICRA'03)*, Taipei, Taiwan, 2003, pp. 2714–2719.

[10] ——, "Coverage, exploration and deployment by a mobile robot and communication network," in *The 2nd International Workshop on Information Processing in Sensor Networks (IPSN '03)*, Palo Alto, 2003, pp. 376–391.

[11] J. Elson, "Time synchronization in wireless sensor networks," Ph.D. dissertation, University of California, Los Angeles, May 2003.

[12] Q. Li, M. DeRosa, and D. Rus, "Distributed algorithms for guiding navigation across a sensor network," Dartmouth, Computer Science Technical Report, Tech. Rep. TR2002-435, October 2002.

[13] D. J. White, *Markov Decision Process*. West Sussex, England: John Wiley & Sons, 1993.

[14] S. Koenig and R. G. Simmons, "Complexity analysis of real-time reinforcement learning applied to finding shortest paths in deterministic domains," Carnegie Mellon University, School of Computer Science, Carnegie Mellon University, Pittsburg, PA 15213, Tech. Rep. CMU-CS-93-106, December 1992.

[15] K. S. J. Pister, J. M. Kahn, and B. E. Boser, "Smart dust: Wireless networks of millimeter-scale sensor nodes," *Electronics Research Laboratory Research Summary*, 1999.