Ann Math Artif Intell (2008) 52:229–255
DOI 10.1007/s10472-009-9124-y

Distributed multirobot exploration, mapping, and task allocation

Regis Vincent · Dieter Fox · Jonathan Ko · Kurt Konolige · Benson Limketkai · Benoit Morisset · Charles Ortiz · Dirk Schulz · Benjamin Stewart

Published online: 18 March 2009

Abstract We present an integrated approach to multirobot exploration, mapping and searching suitable for large teams of robots operating in unknown areas lacking an existing supporting communications infrastructure. We present a set of algorithms that have been both implemented and experimentally verified on teams—of what we refer to as *Centibots*—consisting of as many as 100 robots. The results that we present involve search tasks that can be divided into a mapping stage in which robots must jointly explore a large unknown area with the goal of generating a consistent map from the fragment, a search stage in which robots are deployed within the environment in order to systematically search for an object of interest, and a protection phase in which robots are distributed to track any intruders in the search area. During the first stage, the robots actively seek to verify their relative locations in order to ensure consistency when combining data into shared maps; they must also coordinate their exploration strategies so as to maximize the efficiency of exploration. In the second and third stages, robots allocate search tasks among themselves; since tasks are not defined a priori, the robots first produce a topological graph of the area of interest and then generate a set of tasks that reflect spatial and communication constraints. Our system was evaluated under extremely realistic

R. Vincent (✉) · K. Konolige · B. Morisset · C. Ortiz
Artificial Intelligence Center, SRI International, Menlo Park, CA 94025, USA
e-mail: vincent@ai.sri.com

D. Fox · J. Ko · B. Limketkai · B. Stewart
Department of Computer Science & Engineering, University of Washington, Seattle, WA 98195, USA

D. Schulz
Department of Computer Science III, University of Bonn, Bonn, Germany

real-world conditions. An outside evaluation team found the system to be highly efficient and robust.

Keywords Distributed Exploration · Distributed Mapping · Task allocation · Robots

Mathematics Subject Classifications (2000) 90C35 · 15A90 · 90C30

1 Introduction

Efficient exploration of unknown environments is a fundamental problem in mobile robotics. Increasing efficiency is one of the key reasons for deploying teams of robots instead of single robots. Compared to the problems occurring in single robot exploration, the extension to multiple robots poses several new challenges, including (1) coordination of robots during exploration, (2) integration of information collected by different robots into a consistent map, (3) dealing with limited communication, and (4) what we will refer to as "task inference" and also task allocation.

Coordination during exploration and mapping As the size of a robot team increases, the problem of coordination between robots can also increase in difficulty. The difficulty of the coordination task strongly depends on a robot's level of knowledge: if the robots know their relative locations and share a map of the area explored so far, then effective coordination can be achieved by guiding the robots into different, non overlapping areas of the environment [3, 4, 39, 48]. This can be done by assigning the robots to different *exploration frontiers*, which are transitions from explored free space to unexplored areas [3, 47]. However, if the robots do not know their relative locations, then it is far less obvious how to effectively coordinate them, since the robots do not share a common map or frame of reference.

Map merging To build a consistent model of an environment, the data collected by each robot must be integrated into a single map. Furthermore, such an integration should be done as early as possible, since the availability of a shared map greatly facilitates the coordination between robots. If the initial locations of the robots are known, map merging is a rather straightforward extension of single robot mapping [9, 25, 42, 46], because the data traces of the individual robots can be treated as if they were collected by a single robot. Consistent integration of data when the robots do not know their relative locations is more difficult, since it is not clear how and where the robots' traces should be connected.

Limited communication During exploration of large-scale environments, communication between robots and a control station might fail. To achieve robustness against such failures, each robot must be able to continue exploration on its own, i.e., without guidance by a central control node. Furthermore, groups of robots should be able to coordinate their actions without the need of a central control node, and each robot should be able to take over the task of coordination.

Task inference and allocation The sorts of target problems we have discussed so far differ from conventional multi-agent task allocation problems in which the set of tasks to be allocated within a team is known *a priori*. Tasks here correspond to sequences of navigating to a particular point, stopping, and rotating 360 degrees to

inspect an area for some object of interest (OOI). Although the mapping and search stages could be combined, in our system they were accomplished using separate resource pools. Since one of the goals of this project was to study the distributed coordination of as many as one hundred robots, the cost of the laser sensors necessary for mapping would have been prohibitive if we had tried to replicate the mapping capability on *each* robot. We had two kinds of robots, the mapper are equipped with laser range finder only can do mapping but can not detect the OOI. The other robots have sonars and stationary cameras and need the map to move in the environment and can detect the OOI. The group task is to search a large area systematically for the OOI. However, any spatial area can be partitioned into sub areas in an infinite number of ways; furthermore, the spatial layout of the area is not known ahead of time. Consequently, the robots must decide how to divide up the area in the most effective manner to support the search process. This group task is complicated by the requirement that communications back to a command center (to report back with results) be maintained. We refer to the problem of generating the task list as the problem of *task inference*, which precedes the task allocation process. The latter is one that must be supported on a persistent basis: that is, if a robot runs out of battery power or breaks down, its task should be reallocated among teammates (Fig. 1).

In this paper we present an integrated multirobot system that addresses all of these challenges. The approach is suitable for multirobot search problems that can be

Fig. 1 CentiBots development and evaluation team. The four larger robots in the back, Pioneer IIs with SICK laser range-finders, are used for mapping and exploration. The rest are Amigobots with sonars, a camera and a small PC on top

divided into the following three stages: map building, task allocation for searching for an object, and area protection. Our system was evaluated thoroughly by an outside evaluation team. The results of the evaluations demonstrated that our approach was highly efficient and robust. The maps generated by the team of robots are consistently more accurate than those generated by manual measurements of the locations and extensions of rooms and objects.

This paper is organized as follows. In the next section, we provide an overview of our multirobot coordinated mapping technique. Then, in Section 3, we show how the data collected by multiple robots can be integrated into a consistent map of an environment. Finally, we discuss the task inference and allocation process followed by a description of the experiments supporting the reliability of our techniques. We conclude in Section 6.

2 Decision-theoretic coordinated mapping

We will now discuss the concept underlying our multirobot coordination technique; implementation details will be provided in the experimental results described in Section 5.

2.1 Related work

Virtually all existing approaches to coordinated multirobot exploration assume that all robots know their locations in a shared (partial) map of the environment. Using such a map, effective coordination can be achieved by extracting *exploration frontiers* from the partial map and assigning robots to frontiers based on a global measure of quality [3, 4, 39, 41, 48]. As illustrated in Fig. 2, exploration frontiers are borders of

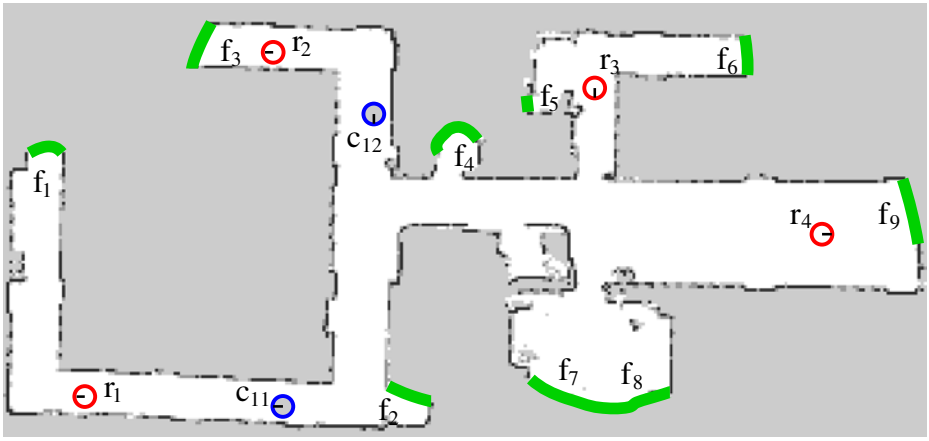


Fig. 2 Coordination example: Partial map built by exploration cluster of four robots (circles $r_1, \ldots, r_4$). Additionally, two location hypotheses (circles c_{11}, c_{21}) have been generated for robot c_1, which is not yet part of the cluster. The map has nine exploration frontiers ($f_1, \ldots, f_9$), indicated by gray lines

the partial map at which explored free space is next to unexplored areas [3, 31, 47]. These borders thus represent locations that are reachable from within the partial map *and* provide opportunities for exploring unknown terrain, thereby allowing the robots to greedily maximize information gain [22, 44]. As a measure of the quality of an assignment of robots to frontiers, the overall travel distance combined with an estimate of the unexplored area at each frontier proved to be a highly measure successful in practice [3, 39].

The assumption of the availability of a shared map, however, severely restricts the scenarios that can be handled by such an exploration strategy. For instance, a unique, globally consistent map can be generated only if the robots know their relative locations. If the robots do not know their relative locations, then it is not clear how they can combine their maps into a global, shared map. Knowledge about relative locations is readily available only if all robots start at the same location or have sensors that provide location estimates in a global frame of reference. While the latter case can hold when using GPS for outdoor exploration [33], there exists no global positioning sensor for indoor environments. Thus, to deal with more general exploration settings, the robots must be able to handle uncertainty in their relative locations, which directly translates into uncertainty in how to combine their maps.

In a full Bayesian treatment, the robots could estimate posterior probability distributions over their relative locations and then coordinate their actions based on the resulting distribution over shared maps. While such an approach could lead to a highly effective exploration strategy, it does not scale well since the number of possible relative locations, and thus maps, grows exponentially in the number of robots. To avoid this complexity, virtually all approaches to multirobot mapping under position uncertainty let the robots explore independently until they have reliable estimates of their relative location; at which time their maps are merged and the robots begin to coordinate exploration strategies [5, 9, 19, 21, 42, 46]. To estimate relative locations, Howard and colleagues rely on the robots' ability to detect each other [19]. Here, all robots independently explore until one coincidentally detects another robot. During such encounters, robots are able to determine their relative location and combine maps. While such an approach scales well in the number of robots, it can result in inefficient exploration, since it can take arbitrarily long until robots coincidentally detect each other. For instance, if one robot follows the path of another robot without knowing, both robots might explore the complete map without ever detecting each other. Other approaches establish relative locations between pairs of robots by estimating one robot's location in another robot's map. This is typically done under the assumption that one robot starts in the map already built by the other robot [9, 42, 46] or that there exists an overlap between the partial maps [5]. Since these techniques do not verify location estimates, they might erroneously merge maps, which typically results in inconsistent maps.

Our approach combines and extends these ideas in order to generate an efficient and robust exploration system. In contrast to [5, 9, 42, 46], our techniques make *no assumptions about the relative locations of robots*. Furthermore, our approach adds robustness by *verifying hypotheses* for the relative location of robots. Similar to [19], this is done by making use of information obtained when one robot detects another. However, in contrast to [19], these detections are not coincidental; they are pursued *actively*.

2.2 Decision-theoretic coordination

Our technique for exploration with unknown start locations integrates information obtained during robot detections into a Bayesian, decision-theoretic exploration strategy. Our system works as follows. Initially, the robots might not know their relative locations: each robot will explore on its own, mapping an increasingly large portion of the environment. As soon as two robots can communicate, they begin to exchange sensor data and estimate their relative location. Once they have a good hypothesis for their relative location, they actively verify this hypothesis using a rendez-vous technique. If successful, the robots form an exploration cluster: they combine their data into a shared map and start to coordinate their exploration actions. If the relative location hypothesis turns out to be incorrect, the robots continue to explore independently and exchange sensor data so as to refine the estimates of relative location. During exploration, the size of exploration clusters increases as more robots determine their relative locations, ending in a single cluster containing all robots.

As long as a robot is not part of an exploration cluster, it individually explores an environment by moving to the closest exploration frontier in its partial map [22, 47]. To coordinate the robots within an exploration cluster, we extend the decision-theoretic approaches of [3, 4, 39, 48] to the case of relative position uncertainty [21]. To do so, we assume that the robots within an exploration cluster share a map and that the positions r_i of all robots in the shared map are known. Figure 2 shows an exploration cluster of four robots sharing a partial occupancy grid map. Exploration frontiers f_i are indicated by thick green lines. The figure also shows hypotheses c_{11} and c_{21} for the location of a robot not yet part of the cluster. In general, let c_{ij} denote the i-th hypothesis for the unknown location of robot j and $p(c_{ij})$ is the probability that robot j actually is at this hypothesized location (how hypotheses and their probabilities are determined will be described in Section 3.1).

The robots in an exploration cluster trade off exploring unknown terrain and verifying hypotheses for the locations of other robots. Hypothesis verification is done by sending one of the robots to the hypothesized location and attempting to physically sense the other robot. In our system, similar to [14], robot detections are performed by marking robots with highly reflective tape and using laser range finders to detect the markers. Once a location hypothesis is verified, the data of this robot can be added to the cluster map and the robot can participate in coordinated exploration. At any time, each robot in the exploration cluster can be assigned either to an exploration frontier or to a hypothesized location of a robot outside the cluster. Coordination between robots corresponds to the the problem of finding an assignment from robots to frontiers and hypotheses that maximize a utility-cost trade-off. To see how this is done, let θ denote an assignment that determines which robot should move to which target (frontiers and hypotheses). Each robot is assigned to exactly one target and $\theta(i, j) = 1$ if the i-th robot in the exploration cluster is assigned to the j-th target. Among all assignments we choose the one that maximizes expected utility minus expected cost:

$$\theta^* = \underset{\theta}{\operatorname{argmax}} \sum_{(i,j)\in\theta} \theta(i, j)\,(\mathrm{U}(i, j) - \mathrm{C}(i, j)) \tag{1}$$

The cost and utility of each robot target pair (i, j) can be computed as follows.

Cost If the target is a frontier then the cost is given by the minimum cost path from the robot's position r_i to the frontier position f_k. Minimal cost paths can be computed efficiently by A^* search. For hypothesis verification, the cost is given by the minimal path to a meeting point between the robots plus the cost to establish whether the two robots actually meet or not.

$$\mathrm{C}(i, j) = \begin{cases} \mathrm{dist}(r_i, f_k) & \text{if } j\text{-th target is frontier } f_k \\ \mathrm{verify}(r_i, c_{pq}) & \text{if } j\text{-th target is hypothesis } c_{pq} \end{cases} \tag{2}$$

Utilities If the target is a frontier, then the utility is given by the expected area that the robot will explore at that frontier. That area is estimated using an estimate of the size of the unknown area visible from the frontier [3]. If the target is a location hypothesis, say c_{pq}, then the utility is given by the expected utility of meeting robot r_q. The function *coord* estimates this utility by measuring the map size of the other robot plus the expected utility of coordinated exploration versus independent exploration. Since it is not known whether the other robot is at the location hypothesis, the utility of meeting is weighted by the probability of the hypothesis, denoted $p(c_{pq})$.

$$\mathrm{U}(i, j) = \begin{cases} \mathrm{explore}(r_i, f_k) & \text{if } j\text{-th target is frontier } f_k \\ p(c_{pq})\mathrm{coord}(r_q) & \text{if } j\text{-th target is hypothesis } c_{pq} \end{cases} \tag{3}$$

Once the pairwise utilities and costs are computed, we use a linear program solver to find the optimal assignment. Finding optimal assignments can be performed in $O(mn)$ time, where m is the number of robots and n is the number of goals [15]. In exploration scenarios involving as many as six robots, we found the overall computation time for this decision step to be negligible compared to the cost of exploration (less than 1 second). Using the trade-off between Eq. 2 and Eq. 3, robots typically move to exploration frontiers and choose a hypothesis as a target only if it is not too far away and its probability is very high.

3 Multirobot map merging

We will now describe how to build a consistent map from data collected by multiple robots.

3.1 Estimating relative positions

Before combining multirobot data into a global map, one has to determine the relative locations of the robots. We now briefly discuss our algorithm for sequentially estimating the relative locations between pairs of robots; more details can be found in [12]. To perform this estimation, robots exchange laser range scans and odometry motion information whenever they are in communication range. Our approach considers only pairs of robots since the complexity of estimating relative locations is exponential in the number of robots considered jointly.

We use an adapted particle filter in combination with a predictive model of indoor environments to determine whether and how the partial maps of two robots overlap. Existing approaches to robot localization have addressed the problem of localizing a robot only in a complete map of an environment. Particle filters have been applied

with great success to this problem [10, 11, 20, 28]. The main difference between localizing a robot in a complete map and in a partial map of an environment is that in the latter case a robot might not be inside the partial map, which it can enter or exit at any time. To deal with this problem, the particles in our approach represent entire robot trajectories, similar to the application of Rao-Blackwellised particle filters for mobile robot mapping [8, 18, 30]. These trajectories are extended whenever new odometry and sensor information becomes available. Furthermore, at every time step, new trajectories are initialized at the entry points, or frontiers, of the partial map. These trajectories represent the possibility that the other robot might enter the partial map at any time. Using a predictive model for observations outside the partial map [14, 40], our approach additionally updates the probability of whether or not the other robot is currently outside the partial map.

At each iteration of the particle filter, hypotheses for the location of a robot are extracted from the sample set and then used by the decision-theoretic coordination technique described in Section 2.2. Once the coordination approach considers a hypothesis valuable enough, it verifies this hypothesis by assigning it to a robot. If this robot detects the other robot at the hypothesized position, its data can be merged into the cluster map, as described next. If, however, the hypothesis turns out to be incorrect, then the particle filter naturally incorporates such information by giving the samples at the wrongly hypothesized location extremely low weights. The low weights result in the removal of these particles in the next resampling step, thereby increasing the probability of alternative hypotheses.

3.2 SLAM paradigms and local maps

The key problem in mobile robot mapping is caused by the uncertainty in a robot's position as it explores an environment. This position uncertainty has to be considered when generating a map from the observations made by the robot. It is this connection between robot position and map uncertainty that makes the Simultaneous Localization and Mapping (SLAM) problem computationally demanding [6, 43]. Over the last years, various research groups have developed efficient solutions to the SLAM problem. These techniques range from splitting maps into submaps [34], to thin junction-tree approximations [36], to sparse extended information filters [43], to Rao-Blackwellised particle filters [8, 18, 30, 32], to graph structures modeling spatial constraints [17, 23, 29]. In this project, we build on the latter class of techniques, which are appropriate because they can be made to be independent of the coordinate system in which the constraints are expressed [24], an obvious advantage when combining local maps that have different coordinate systems. Here, we provide only an intuitive description of our approach; more details can be found in [17, 24, 29]; a good exposition of general constraint graphs can be found in the Graph-SLAM algorithm [44].

3.3 Map merging examples

The constraint graph is ideal for integrating map information with uncertain alignment. In the case of odometry and local scan matches, the system looks at just a small local neighborhood to enforce consistency [17], which can be done in constant time. The more interesting cases are enforcing global consistency involving loop closure in

a local map, and partial map merging. In loop closure, a robot is building a local map using its own scans and the scans of any colocated robots. At some point, the robot returns to a position it has previously visited, but accumulated error can cause it to be misaligned (Fig. 3 left). Here the robot has traversed an interrupted loop, moving out of the top of the figure before coming back. Once scan matching establishes links with poses at the beginning of the loop, additional constraints can be added to the graph. Based on the new constraints, the mapping algorithm determines the optimal position for all scan locations by maximizing the posterior probability of all constraints in the graph (see [13]). In practice, the initial solution established by enforcing local constraints gives a "close enough" solution to start the minimization process. On the right side of Fig. 3, scan matching has established links with poses at the beginning of the loop, resulting in a consistent map after minimization of the constraint system. Because the optimization is efficient, it can be performed online as the robot explores an environment, causing no more than a second or so of hesitation as consistency is enforced.

The constraint representation naturally facilitates the merging of partial maps built by different robots. For example, the upper panels of Fig. 4 show three partial maps built by three robots. Suppose we can link the pose marked "o" in the left map to the pose marked "o" in the middle map, and the poses marked "x" in the middle and the right maps. Then, the three maps can be registered in the same metric space. This is done by taking each constraint in the middle map and adding it to the constraint graph underlying the left map, just as if all scans were collected by a single robot. In addition, we generate an initial solution as input to the global optimization, by transforming all the poses in the middle map, and making a rigid transformation so that they line up with the colocated pose at its correct position. At this point, although the maps are aligned correctly for the colocated poses, they can differ on poses that are distant from this point—see the lower right panel in Fig. 4. An additional "zippering" process is performed, in which all the poses that are now close in the colocated two partial maps are scan-matched for additional constraints. By consolidating the poses into spatial buckets, this process can take place in order N, the number of poses in the partial map. Finally, the scans observed by the third robot are added to this map, using the same process. The occupancy grid map resulting from optimizing the global constraint graph is shown in the lower left panel in Fig. 4.

Abstractly, the zippering process lets us take any partial maps produced by robots and put them together, once a common location (*colocation*) between their trajectories has been identified. In our system, colocation information is estimated by the particle filter described in Section 3.1, and verified via actively triggered robot

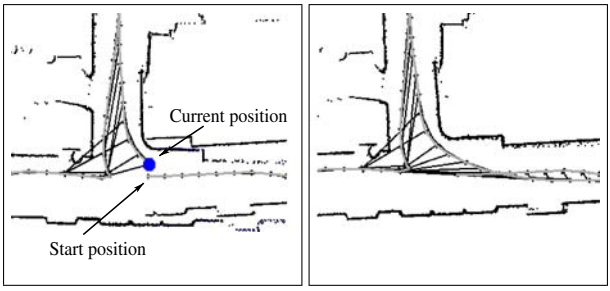


Fig. 3 Pose constraints before (*left*) and after (*right*) linking the start and end of a loop. Minimization of the constraints after the robot returned into the hallway to the right results in consistent scan locations. Robot trajectory is shown in *gray*, spatial constraints as *thin black lines* attached to the trajectory

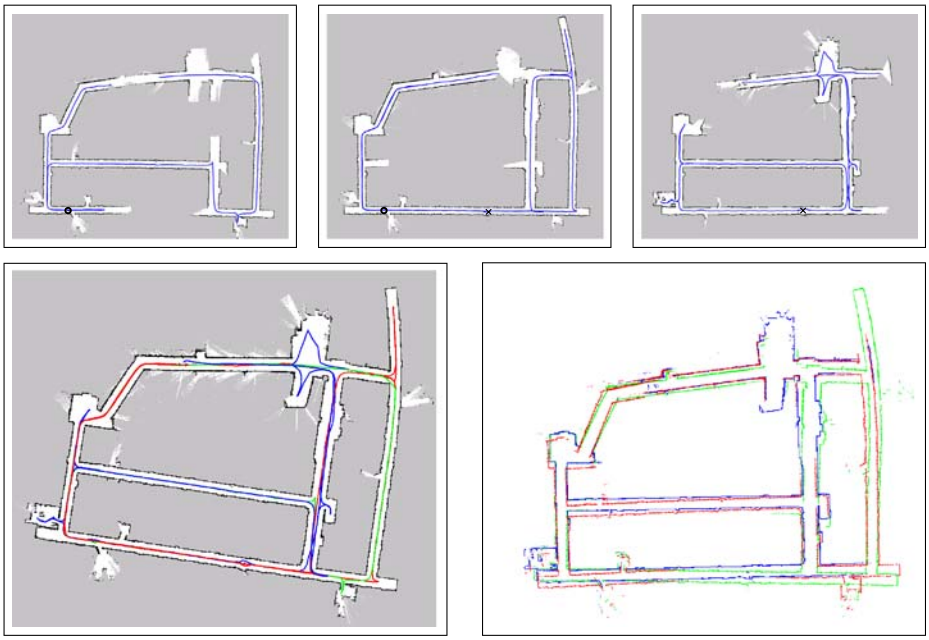


Fig. 4 *Upper panels*: partial maps built by three robots in the UW Allen Center. The 'o's and 'x's provide connection points between the *left* and the *middle map*, and the *middle* and the *right map*, respectively. *Lower panels*: the *left picture* shows the map generated from the three partial maps by optimization of the global constraint graph generated by "zippering" the maps together at the connection points (*right*). Map generated by simply overlaying the partial maps, without any additional global optimization (only laser scans are shown for clarity)

detections, as described in Section 2. Our map merging technique is transitive in the sense that if robot A knows robot B's location inside its partial map and robot B knows the location of robot C inside its partial map, then it is possible to consistently merge C's map into A's map (possibly after merging B's map into A's map). The reader may notice that merging maps in different orders might lead to slightly different maps, which is due to the approximations performed by our approach (sequentially adding spatial constraints might result in different constraint systems). In practice, however, we found this approach to map merging highly reliable.

4 Search

4.1 Task inference

The *primitive actions* available to a Centibot are of three types: perceptual, communication, and motion. Examples include turning a sensor on or off, sending a message, and moving to or rotating around a particular point. In addition to primitive action types, there are higher-level action types that can be constructed from the set of primitives [27].

At the highest level, Centibots' missions are defined in terms of the sequence of three high-level stages discussed earlier, over some spatial area of interest. For

any mission, the system must identify the tasks to be performed by the team for the mission to succeed. As part of the Centibots system, we developed the SPAtial REasoning (SPARE) that identifies tasks as follows. Given the continuous nature of the space, the first step, following the mapping stage, involves a process of *task inference*, T. Let TG stand for what we will call a *topological graph*, S to the map produced in the first phase, R to the resource pool, N to a set of task nodes, and E to a set of edges connecting elements of N. Then, T is a mapping, $T : S \times R \rightarrow TG = \langle N, E \rangle$, where the set R conditions the set of task nodes according to the capabilities reflected in the resource pool (in terms of the number of available resources and types of sensors). Once a TG is computed, the distributed dispatcher, discussed in the next section, allocates resources to each task.

At execution time, individual robots expand the instantiated act types to an executable form. For example, if the robot is committed to `search-for(robot32,OOI, N85)`, it will expand that act type to a sequence of movements (after computing a suitable navigation path) to point N85, stop, and then rotate 360 degrees to search for the OOI. When executed, act types are commonly referred to as *behaviors* [1]. Each robot can also combine act types at execution time, corresponding to blended behaviors [38]. As we will describe, the task inference process computes such combinations by balancing several constraints.

1) *Spatial representation in SPARE* The task inference process operates in a number of steps. First the occupancy map (Fig. 5) produced during the first mission stage is converted into a TG, such that each vertex of the TG corresponds to a goal to cover to perform the OOI search, given the sensor capabilities, and the protection task is reduced to distributing the robots in such a way that each node in N represents a potentially interesting point to be covered.

Several constraints are imposed on each $n \in N$: (1) *clearance*: each n must be a valid position in free space; (2) *reachability*: a valid trajectory must exist for any pair of n's; (3) *completeness*: the entire area must be covered by some n; and (4) *time*: task execution time must be real time.

The algorithm computes the TG by first building a skeleton (Fig. 6) giving the structure of the free space and then generating a TG from the skeleton such that each vertex corresponds to a valid goal and each transition corresponds to a valid trajectory. The skeleton is represented as a *Voronoi Diagram*, which is a collection of polygons (regions) generated by a set of points (sites) such that any point inside one of the regions is closer to that region's site than to any other site. The Vornoi Diagram is computed by wavefront expansion in the discretized map [2]. The complexity of this process is linear with the number of cells in the grid, independent

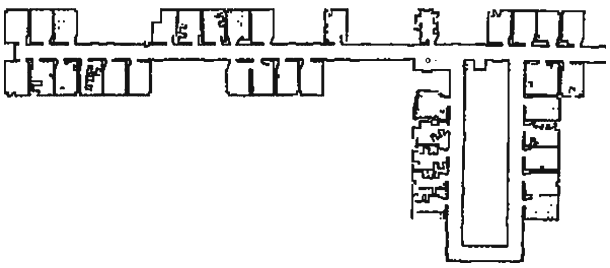


Fig. 5 Occupancy map produced by first wave of mapper robots

Fig. 6 Part of the skeleton

of the shape of obstacles in the environment. The process consists of the following steps: (1) a wavefront is propagated from some sites ("obstacle" cells in the bitmap), and (2) a distance d gives the minimum distance between two sites, the cells at the meeting of two or more waves belonging to the Voronoi Diagram.

From the skeleton we proceed as follows:

1) Vertex identification: cells with one edge or more than three edges are recorded.
2) Edge building between two vertices: the Voronoi components linking the vertices are followed. The length of the component is recorded as a label for the edge.
3) Graph filtering and simplification: redundant vertices are merged followed by a reachability filtering process based on the closest distance recorded in each cell of the diagram.
4) Spatial information: for each edge, we associate the corresponding component of the Voronoi diagram

An additional step identifies the topological type (Fig. 7) of an area (i.e., whether an area corresponds to an office or corridor). This is needed for navigation robustness (speed and sonar parameters must be adjusted when entering a narrow area such as an office) and for performing the searching and intrusion detection. For each edge, heuristic determination of the topological type is based on the closest distance to obstacles and the topological type of the previously classified edges in the neighborhood. Once this process is completed, the topological type (room or corridor) is added to the label of each edge (Fig. 8).

The next step considers the set of all possible assignments, α, from nodes of TG to 1 or 0, depending on whether or not a robot is present at a node. We introduce a cost function, $C(\alpha)$, which is a scalar function of the assignment. Since the number of assignments is exponential in the number of nodes, we decompose the cost into subcosts that can be easily calculated, and use an approximate method to determine a

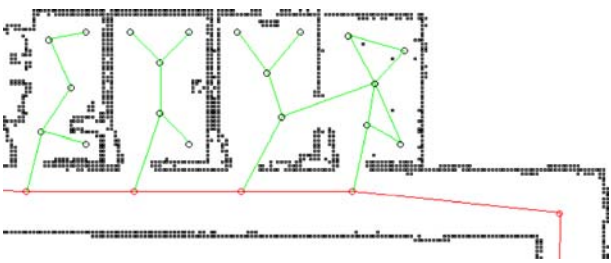


Fig. 7 Part of the topological graph

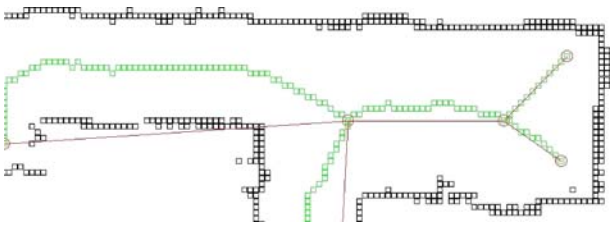


Fig. 8 Skeleton components associated with edges

good assignment. In general, we want costs to be local to a single node, or at least to a small neighborhood of nodes, so that incremental optimization algorithms will work well. To this end, we determine the global cost by summing smaller cost functions:

$$C(\alpha) = \sum_{i=0}^{p} w_i \sum_{j=0}^{n} c_i(v_j, \alpha)$$

where n is the number of nodes in TG, and p is the number of cost functions. The w_i are weights that can be changed to reflect the type of mission under consideration. The weights were chosen empirically, to reflect the different priorities in the two mission stages of searching for the OOI and protecting the OOI.

Note that, potentially, each local cost function c_i could involve the whole assignment. In practice, there is only one such cost function, which is the difference between the number of robots in the assignment, and the desired number of robots for the mission. This cost is easily computed. We have chosen the following ten local cost functions, by observing an expert choose assignment solutions and explain the basis for each.

- c_1: *Corridor occupancy.* From TG, we are able to distinguish corridor areas from office areas. c_1 reflects the corridor occupancy. This function is equal to -1 if a robot is allocated on a vertex with a type "corridor", 0 otherwise.
- c_2: *Office occupancy.* In the same way as c_1, c_2 reflects office occupancy.
- c_3: *Corridor only.* If c_1 favors a robot allocation in a corridor it does not prevent allocations in offices. To have a more exclusive allocation in corridors c_3 returns -1 if the allocated vertex is a type "corridor", 1 otherwise.
- c_4: *Offices only.* In the same way c_4 is added to prevent allocations in corridors. Notice that c_3 and c_4 jointly exclude the allocation of robots to offices.
- c_5: *Sensor coverage*: the distance between two robots should not exceed the maximal sensor range R to assure a consistent coverage of the environment. To evaluate c_5, the shortest distance between any pair of vertices is precomputed using Johnson's algorithm. This computation is done just after the construction of TG. Each time a robot is allocated a vertex v, the distance d to the closest allocated vertex v' to v is computed. The value of c_5 depends on d and R: an excessive overlapping or too large of a distance between v and v' is penalized by c_5. The difference $d - R$ corresponds to discretized intervals. A specific cost is returned for each interval.
- c_6: *Communication coverage:* The position of the OOI must be communicated to the command center by the robot that found the OOI. The range of wireless communication is limited. A backbone between the OOI and the command center must guarantee such communication. To compute c_6, each

vertex belonging to the shortest path between the OOI and the command center is labeled "BB". c_6 equals -1 if the vertex v is labeled "BB", 1 otherwise.

c_7: *Protection of the OOI:* A strong density of robots must be present in the neighborhood of the OOI. c_7 equals 1 if the distance between the allocated vertex and the position of the object of value exceeds a predefined threshold, -1 otherwise.

c_8: *Visibility* : c_8 favors allocations on points with high visibility. The largest number of edges connected to a vertex (N_{Emax}) is recorded during the construction of TG. c_8 is equal to $N_{Emax} - N_E$ where N_E is the number of edges connected to the allocated vertex.

c_9: *Unique path between two areas of the environment:* To compute c_9, each vertex corresponding to an articulation point in TG is labeled "AP". c_9 equals 0 if the allocated vertex is tagged "AP", 1 otherwise.

c_{10}: *Number of robots:* The number of available robots to perform a task may vary (breakdowns, empty batteries). This number should be controlled during the resource allocation process. At each allocation, the number of allocated robots N_R is updated. c_{10} returns $N_R^* - N_R$ where N_R^* is the desired number of robots.

For each elementary cost function, c_i, the above rules are used to compute the value in the case of an allocation. Similarly, each c_i returns a value in the case of a deallocation (a vertex previously allocated is deallocated).

Each new weight distribution w_i defines a new task for the system. For instance, if we want to send n robots to the corridors to maintain the communication backbone, only w_1, w_3, w_5, and w_9 will have a value different from 0. To send n robots around the object of interest to maintain the backbone communication, only w_5, w_7, and w_9 are considered.

In the context of our work, two main tasks have been implemented and extensively tested. The first is OOI searching. Since this search must be done over the entire environment (and not for a specific topological area), w_1, w_2, w_3, and w_4 are equal to 0. Since the OOI position is still unknown, w_6 is also equal to 0. All the other weights are considered. The second task is OOI protection. All intruders must be detected. In a very large environment, we favor allocations in the corridors to limit the number of robots, sensor coverage, global visibility and backbone communication. Then, to perform this task, only w_2 and w_3 are set to 0.

2)*Task resolution* We want to find the n-tuple $(v_1, v_2, ...v_n)$ optimizing C with a weight distribution w_i specific to the task. We observed that the initial TG does not contain enough vertices to provide good coverage of the environment and to obtain a satisfying solution (no possibility to place a robot along a long edge). To obtain better coverage of the environment, new vertices are, therefore, added to TG along the skeleton component of each edge of TG (Figs. 8 and 9).

The search space associated with the optimization of C has a size 2^n where n is the number of vertices in TM (typically, several hundred). In this huge search space and in the context of our application, our goal is not to find an optimal solution but an approximate one in a reasonable time (within a few minutes). The quality of a solution is determined by human expertise: a solution is considered good if no misallocation is detected by a human analyzing the result.

We used simulated annealing to compute the solution based on a linear cooling schedule ($T_{new} = T_{old} - dt$), where T is the temperature. The algorithm starts from

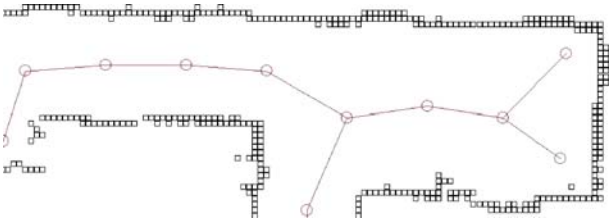


Fig. 9 Addition of vertices along each component

an initial random allocation respecting the desired number of robots if w_{10} is not null. While T is greater than 0, n vertices v_i of TG are picked randomly. For each v_i, C_i is the current cost locally associated to v_i. If v_i is allocated (resp. not allocated), C'_i is the new cost computed by deallocating v_i (resp. allocating v_i). If $\delta C = C'_i - C_i < 0$, the allocation of v_i is changed. If $\delta C \geq 0$, the probability p to change the allocation of v_i depends on T and is computed with $p = \exp(-\delta C/T)$. If the allocation of v_i has been changed, C_i is updated. T is then updated and another cycle is started.

We also tried approaches other than simulated annealing but they were not successful. For example, using genetic algorithms the initial population turns out to consist of 10,000 genes, where the number of genes corresponds to the number of vertices in the graph and the value for a gene (1 or 0) depends on whether it is allocated or not allocated. The problem we noted was very slow computation time, and convergence was difficult to control.

Results We generated TGs for environments as large as 24,000 square feet. The size of the TG in such cases was approximately 500 nodes. During the search task, the cost functions used were c_5, c_8, c_{10}; a random solution had an average cost of 7,950 whereas our SPARE system was able to compute a solution with cost 3,200 with an average CPU time of 50 ms. For the protection task, the cost functions used were $c_1, c_5, c_6, c_7, c_8, c_9, c_{10}$. The average cost of a random solution was 11,530; the average cost of a SPARE solution was −1,540, with an average CPU time of 200 ms on a Pentium III 1 Ghz.

4.2 Distributed task allocation

Having identified the tasks that need to be performed, the robots are then assigned to tasks and commitments secured for each robot using a process that we call "dispatching". Since the Centibots system is intended to operate over multiple robot power cycles, the dispatcher must also be responsible for reallocating resources. The system makes extensive use of the Jini [7] architecture. Each robot and each algorithm is a network service that registers, advertises, and interacts independently of physical location. Services include a map publisher that aggregates data from mappers and publishes the map to other robots, the dispatcher that allocates tasks to robots, and the user interface. The result is a very modular, scalable infrastructure.

Each robot is completely autonomous and is able to reach any node in the map by using its own local path planner [26]. The metaphor we use to describe the dispatcher system is that of a taxi dispatcher, where each robot corresponds to an independent taxi. There are two modes of operation; one is managed and the second is auction based. In the managed mode, when a robot (taxi) is ready to work, it informs

the dispatcher by communicating its position and battery level; the dispatcher then assigns the taxi one fare (in our case, a node to navigate to and execute a predefined behavior). In the auction-based mode, when a robot (taxi) is ready to work, it asks for the list of jobs available and ranks them using its own preference function (e.g., current position, battery level). Once ranked, the robot (taxi) bids with its ranked job list and the dispatcher allocates jobs in preference order. Subsection 5.5 explains in more detail the differences in the two modes of operation.

4.3 Team-oriented behavior in Centibots

The question of whether a group of agents is truly collaborating on a task is a difficult one. Rich belief-desire-intention (BDI) theories, such as SharedPlans, model the constraints on the evolution of mental states necessary for collaboration [16]. Within a real-time robotic system, explicitly representing and reasoning with such complex theories is a major challenge. Given the limited computational resources available on the sorts of simple robots that make up the Centibots system, it is more reasonable to use such theories as "blueprints" for design; such an approach is one advocated in [16], and is the one that we have adopted. SharedPlans is not the only such theory that could provide such a blueprint, but it is one of the most complete existing theories of collaboration.

According to SharedPlans, the following summarizes the major elements of collaboration involving some task (the methods employed by Centibots to address these requirements are shown in parentheses): (1) A set of agents has been identified for the task (the dispatcher(s) monitors the robots and also runs auctions to identify capable agents); (2) the agents agree on a recipe (currently, this is built in to the behaviors represented and used by each robot within the Saphira system); (3) agents provide help when needed (the system and individual agents make use of an intelligent monitoring system, described elsewhere [45], to identify failures and unexpected events that need to be addressed); (4) agents are committed to the success of the joint activity (again, supported by the monitoring system and by the dispatcher in making use of robot resources to ensure survivability of the team); and (5) agents communicate when they need help (agents incorporate communication constraints when localizing themselves to maintain team communication).

Another view of team collaboration is that explored by team theory [37] in which teamwork is identified with agents that are working to maximize social welfare or team utility. The Centibots system also satisfies this view of collaboration: the construction of TG was based on the maximization of a weighted set of constraints.

5 Experimental evaluation

We will now describe the evaluation of our exploration system. Additional aspects of the approach are evaluated in [14, 21, 23, 40].

5.1 Implementation details

1) *Coordination and mapping*: We implemented the decision-theoretic coordinated mapping technique described in Section 2.2. Maps are represented compactly

as sets of laser range scans annotated with robot poses and probabilistic links (scans are recorded only every 50 cm of translation or 30 degrees of rotation). Within an exploration cluster, each robot integrates its observations into its own map, and broadcasts the information to the other robots. While most of the other robots only store this data, the team leader of the cluster integrates all the sensor information it receives. Thus the team leader, which is chosen as the robot with the smallest ID, has a complete and consistent map representing the data collected by all robots in the cluster. This map is used to coordinate the robots in the cluster. Whenever two clusters meet and merge their maps, the team leader with the smaller ID becomes the leader of the new exploration cluster. Frequently, the team leader broadcasts the map to the other robots to guarantee consistency. This data can be sent very compactly, since only updated robot poses and links have to be transmitted (scans are already stored by the other robots). The most complex broadcast follows whenever a robot closes a loop, since the optimization of the constraint system modifies all robot poses in a map (Section 3). In practice, broadcasting even this information typically involves sending only several kilobytes of data, which is well below the capacity of typical 802.11 wireless communication capacity and can be done in a fraction of a second.

A crucial situation occurs when a robot moves into the communication range of a robot from another, possible single-robot, cluster. At this point in time, the robots exchange all their sensor and motion information and start estimating their relative locations using a particle filter approach, as described in Section 3.1. Our current system allocates this task to the team leader. The other robots in the team do not generate hypotheses. In the worst case, if all robots are in single-robot clusters and within communication range, the number of particle filters run on each robot can be as high as the total number of robots minus one. In simulation experiments, we found this simple approach to work efficiently enough for as many as six robots. However, it does not scale to very large teams of robots and more thought must be put into more intelligent allocation of computation tasks.

2) *Dealing with limited communication*: Our exploration system achieves robustness to communication loss by enabling every robot to explore the environment on its own. Whenever a robot in an exploration cluster reaches an assigned goal point, it keeps on exploring based on its own map until it receives a new goal point. Thus, if a robot moves outside the communication range of its cluster, it automatically keeps on building its own map until it gets back into communication range. After getting back into communication, robots exchange all the relevant data that had not been yet shared because of the communication disruption. Such a "sync" operation only involves the communication of rather small data sets and can typically be done in less than a second. Our approach is also robust to loss of the team leader, since any other robot in the cluster can explore on its own or take over the team leader role. In the extreme, if none of the robots can communicate with each other, each robot will explore the environment independently of the other robots. The result will still be a complete map, only built less efficiently.

We added hand shaking and various timeouts to the decision making in order to make active hypothesis verification robust to loss of communication. This implementation task turned out to be rather tedious since it required extensive

testing of the system to determine all situations in which one robot might leave the communication range of another robot. As an example of our approach, when a robot sends a "Meet" signal to another robot for which it has a good location hypothesis, it waits for an acknowledgment of this signal. If the acknowledgment is not received after several seconds, the robot keeps on exploring and reconsiders a meeting only after an additional timeout and the other robot is back in communication.

5.2 Evaluation experiments

The exploration and mapping system that we have described was evaluated thoroughly as part of the Centibots project (involving SRI International, the University of Washington, and ActiveMedia Robotics) within the DARPA Software for Distributed Robotics (SDR) program. The SDR program was unique in having an experimental validation conducted by an outside group. For a week in January 2004, the CentiBots were tested at a 650 m^2 building in Ft. A.P. Hill, Virginia. Testing was done under controlled conditions, with a single operator in charge of the robot team. All computation was performed using state-of-the-art laptops onboard the robots. The evaluation criteria for mapping included time to create a map, topological accuracy, and percent of area mapped. Ground truth for mapping was given by a manually constructed map (Fig. 10d). Extensive software tuning was circumvented

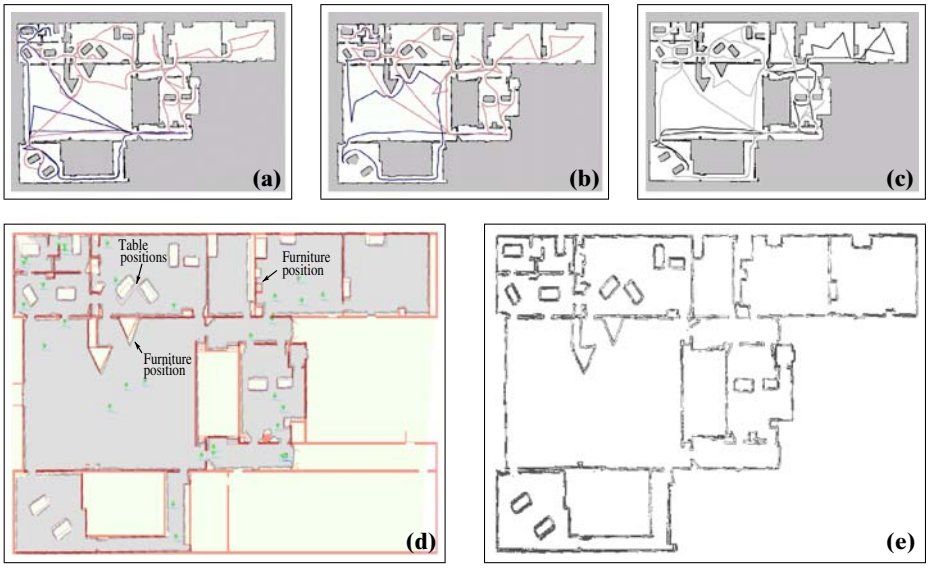


Fig. 10 **a**–**c** Maps built during three autonomous exploration runs. The maps look almost identical, even though they were built under very different circumstances. The similarity between the maps illustrates the robustness of the system and supports our belief that these maps are more accurate than the hand-built map. **d** Map overlayed with the ground truth CAD model of the building. The CAD model was generated by manually measuring the locations and extensions of rooms and objects. **e** Map generated from overlaying the three maps shown in **a**–**c**. *White pixels* indicate locations at which all three maps agree, black pixels show disagreement on occupancy

by limiting access to only half of the experimental area during test runs. In addition, the developer team was not allowed to visually inspect the complete environment before the robots were deployed.

The results for five official mapping runs are summarized in Table 1. In all runs, the robots autonomously generated a highly accurate map of the environment. The average mapping time for single robot exploration was 24 min; this time was reduced to 18 min when using two and 15 min when using three robots. It should be noted that the robots frequently lost communication with the overall control center and with other robots. In such situations, the robots explored on their own and combined their sensor data as soon as they were in contact again.

In addition to demonstrating the robustness of our system, an important result of this evaluation was the fact that the maps generated by the robot team were more accurate than those built manually by the evaluation team. Figure 10d shows one of our maps overlayed with the "ground truth" map. As can be seen, the two maps do not match perfectly (see for instance the two tables in the upper middle room). Three maps built by our robots in three different evaluation runs are shown in Fig. 10a–c. These maps look virtually identical, even though they were built independently of each other, using different trajectories of the robots. To better illustrate the similarity of these three maps, we overlayed them on top of each other. Figure 10e shows the pixels at which the overlayed occupancy grid maps are not identical. As can be seen, the maps almost perfectly line up. Mismatches are only along the obstacles, which is mostly due to the limited resolution of the maps.

5.3 University of Washington Allen Center Evaluations

We performed additional evaluation runs with three robots in the University of Washington Allen Center. These runs confirmed the reliability of our system. Figure 12 illustrates one of these runs. An animation of this run can be found at http://www.cs.washington.edu/robotics/projects/centibots.

To evaluate the benefits of active colocation, we performed several simulation runs involving three robots. We used the Saphira robot simulator along with a map of the first floor of the Allen Center (see Fig. 11). The Saphira simulator accurately models robots including noise in motion and sensing. A larger environment was simulated by limiting the velocity of robots to 20 cm/s and the range of detections to 1.5 m (Fig. 12).

Table 2 summarizes the results of 12 exploration runs, six of which were performed using our active colocation approach (numbers are mean times along with 95% confidence intervals). The other six runs merged maps only when robots met coincidentally, similar to the approach discussed in [19]. Our map-merging technique generated accurate, consistent maps in all 12 runs. As can be seen, actively verifying

Table 1 Exploration runs during the FT. A.P. Hill evaluation

Run	# Mapping robots	Mapping time	Map area
1	1	22 min	96%
2	1	26 min	97%
3	2	17 min	95%
4	2	19 min	96%
5	3	15 min	97%

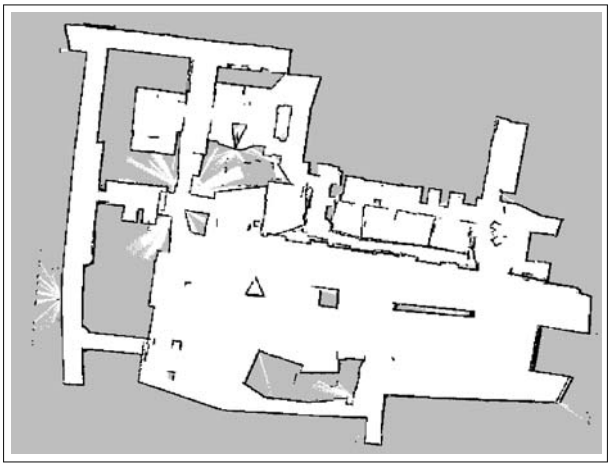


Fig. 11 Map used for simulation experiments

relative location hypotheses significantly reduces the overall exploration time. The third and fourth columns of the table indicate why our active approach is more efficient than its passive counterpart. The third column indicates the time until the first two robots are able to merge their maps, and the fourth column gives the time until the third robot joins the exploration cluster. As can be seen, by actively

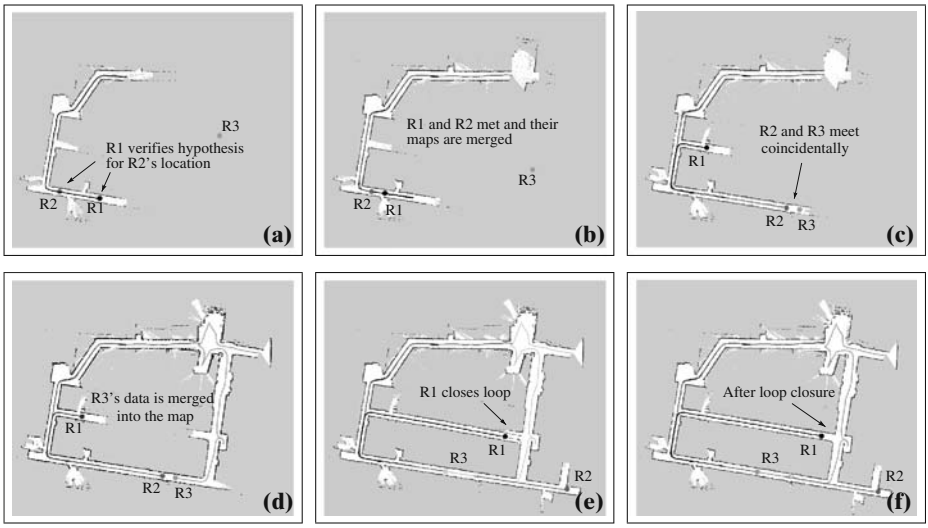


Fig. 12 Sequence of partial maps generated during exploration with three robots. Shown is only the map of robot R1 along with the locations of the other two robots. In this experiment, the robots do not know their relative start locations. **a** R2 is in R1's map and R1 has a high probability hypothesis for R2's location. R1 sends R2 a "Stop" command and decides to verify this hypothesis. **b** After R1 meets R2 at the hypothesized location, the robots merge their maps. They now coordinate their exploration. **c** R2 and R3 meet incidentally and **d** merge R3's data into the map. **e** R1 closes a loop. **f** Since all data is integrated into a global constraint graph, R1 is able to correct the odometry error. The final map of this run is shown in the lower left panel of Fig. 4

Table 2 Exploration with and without active colocation

Approach	Mapping time [min]	First meeting [min]	Second meeting [min]
Passive	35.2±3.8	15.0±4.1	25.2±7.9
Active	26.3±2.4	6.2±1.3	16.3±3.8

verifying hypotheses, the robots merge their maps earlier, which results in improved coordination between their exploration strategies.

We performed further simulation runs in this environment using six robots, which resulted in faster exploration of 22.8±4.5 min. All these runs resulted in globally consistent maps. Furthermore, these runs involved active colocation between exploration clusters of more than one robot each.

5.4 Search experimental results

We experimented with several task allocation strategies. The problem is to minimize the search time, where all robots start from the same position. This problem is in theory similar to a multiple traveling salesman problem with the following differences: there is no *a priori* specification of the number of available salesmen and one can fail at any time during execution. One obvious strategy is to expand the search from the starting point, choosing the closest point from the current location. The second obvious strategy is the opposite: the robots attempt to reach the farthest possible job. The major difference between a real taxi dispatcher and our dispatcher algorithm has to do with the utility and cost functions. In a real taxi operation, the longer the fare is, the more money that is made. In our system, all nodes have equal reward and no cost is incurred. The only metric the system is trying to minimize is the time to complete a search.

We tested the system in three live, realistic experiments monitored by DARPA. One experiment involved a 24,000 square foot area.[1] The SPARE algorithm produced a TG with 499 nodes.

To simulate the environment in which the robots are performing the search, we used a modified version of the Saphira simulator built for the robots. The simulator is capable of simulating all the robot sensors and actuators (laser range finder, sonars, motor controls) using the same programs. In addition, the map that the simulator uses for the environment is the exact map produced by the mapper robots in the first phase of an actual physical run. As far as the search robots are concerned there is no difference between a simulated run and an actual run. We used this simulator to run different task allocation strategies and observe the performance profiles of each of them. The simulation lacks one key feature, which is the ability to simulate collisions with other robots. In all simulations, robots are *transparent* to each other and do not affect each other. This feature is crucial if we had to make performance predictions of an algorithm in the real world but we were only interested in relative performance, in the same setup, among alternative strategies. Figure 13 shows the search completion time for that experiment: the strategy of choosing the closest job to the robot seems to be the best one. When the strategy was actually executed, however, the performance

[1] A video of this experiment is available at the Centibots website: http://www.ai.sri.com/Centibots/pictures/demo2_small_mov.html.

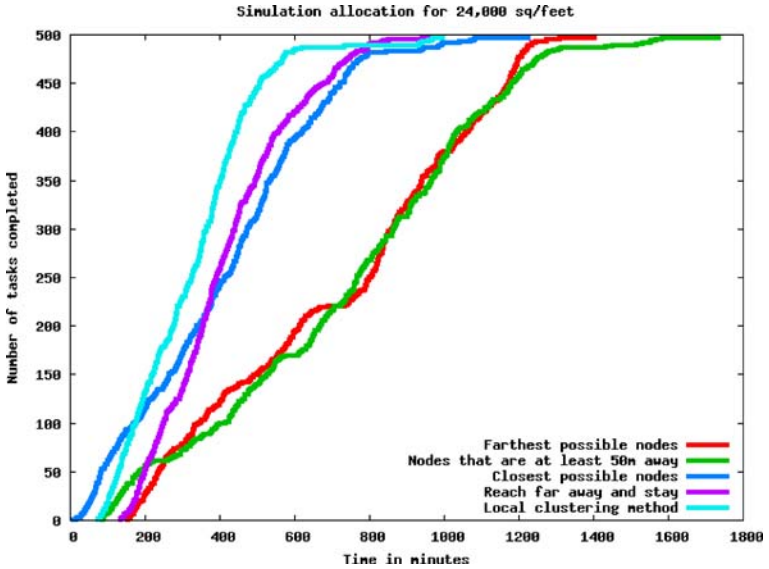


Fig. 13 Comparison of all obvious strategies for task allocation. The X axis represents time and the Y axis represents the number of jobs completed. This data was collected in simulation

was worse than the graph predicted: because of the massive traffic jam created by the robots. One way to avoid the traffic congestion problem is to disperse the group of robots such that a robot will remain in the area originally assigned for any subsequent "local" jobs. This strategy starts like the furthest away strategy for the initial job and then switches to a closest possible strategy for the follow-on jobs; we refer to this strategy as *clustering*. Figure 13 plots the performance of all strategies.

In contrast to the results shown in Fig. 13, which were done in simulation, Fig. 14 illustrates task allocation, using the clustering method, during an official DARPA physical experiment involving 45 robots in which obstacle avoidance and real execution were taken into consideration. As one can see, the system demonstrated similar performance.

Optimizations The physical experiments identified the need for several enhancements to the clustering strategy. The first-needed optimization had to do with the adhoc network employed by the system.[2] The problem is that there is no guarantee of connectivity between teammates when robots begin their tasks or when a robot attempts to contact the dispatcher at the completion of a task to get assigned to a new one. When this happens, a robot will wait for 2 min and then start moving toward its starting point while checking for network connectivity. In this way, the robot eventually reacquires network connectivity. One disadvantage to this approach is that it can result in a great deal of unnecessary travel (and battery usage). We also observed that if the delay is increased (that is, before the robot decides to return to the starting point) other robots would tend to come to the vicinity and provide

[2]We used the Topology Dissemination Based on Reverse-Path Forwarding (TBRPF) protocol [35].

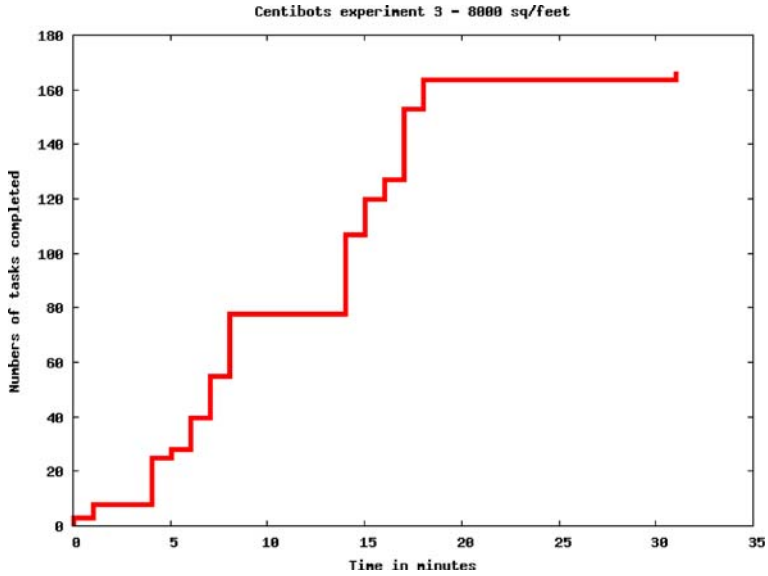
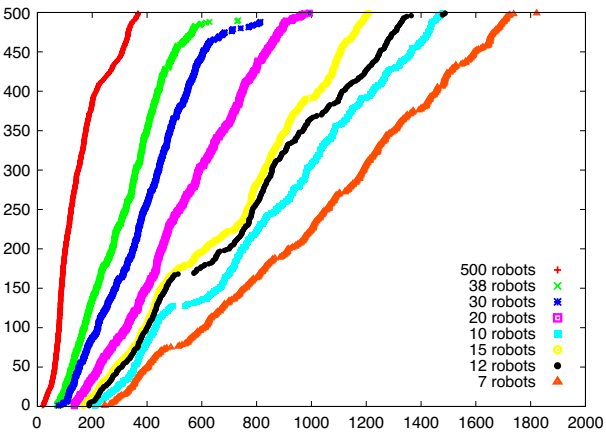


Fig. 14 Graph of task allocation using the clustering method captured during an official experiment using 45 robots.The X axis represents time and the Y axis represents the number of jobs completed

network connectivity. In the search stage, the system tried to minimize search time; an idle robot would, therefore, not result in the most efficient search. We improved the efficiency of idle robots by allowing them to search for *nearby* jobs. This new strategy had several advantages: it simplified the computation of the ranked list of jobs for each robot, it made better use of each robot that traveled outside the current network zone, it decreased the number of messages for the system, and it gained time for the robots to expand the network zone.

Since communication can never be guaranteed, a purely centralized dispatcher has problems associated with it: exchanges with a dispatcher agent can take time. As described earlier, in the Centibots system every algorithm or agent, including the

Fig. 15 Increasing the number of robots available for searching beyond 20 does not yield a faster completion time



dispatcher, is a network service. To avoid reliance on a single centralized dispatcher service, we designed the dispatcher to support hierarchical dispatching. Each robot can register with multiple *dispatching agents*, one of which is considered "preferred". Teams of robots are formed by a *commander*, and for each team, a dispatcher is selected. Each team (robots plus preferred dispatcher) behaves exactly like the single team we have described. The human commander assigns a set of jobs to each team and the teams' dispatchers distribute those tasks to individual robots. When a robot has finished its assigned jobs, it notifies the dispatcher, making itself available, and requests a new set of jobs. If a robot cannot contact its preferred dispatcher or if its preferred dispatcher does not have more jobs available, the robot automatically asks the other dispatchers for jobs. The latter option increases redundancy and reliability while providing a means for load balancing: robots can be reallocated to a dispatcher that has more work than other dispatchers. This load balancing is achieved completely autonomously.

5.5 Auctioneer vs. dispatcher

The Centibots allocation system was designed to allow the dispatcher agent to act as an auctioneer: robots can "bid" on the jobs for which they are the most competent. We found that this feature, however, could introduce networking problems. The robots have access to an 11-Mbps shared network; if, at the start of a search, all 100 robots start communicating, they will saturate the auctioneer and the network, effectively preventing communication (i.e., *network flooding* would occur). As a further complication, all the robots start from the same position, and therefore their bids would all be more or less identical. However, once the robots have started the search and have dispersed, the auctioneer and the network stabilize and produce good performance. Since all the robots are starting from the same position and are executing the same ranking algorithm, one option for avoiding the network flooding problem would be to have the auctioneer compute the robot's expected preferences and then just assign the preferred jobs to the robot. This would decrease the network congestion by reducing the size of messages (robots do not then need to be aware of all of the hundreds of jobs that must be allocated). To implement this option, each robot must periodically pass to the dispatcher/auctioneer its position, battery level, and orientation. However, it turns out that this information is already circulating in the system for use by the user interface: for display purposes, each robot is sending status information to the command control station every second. It was trivial to have each dispatcher/auctioneer eavesdrop on these update messages, and collect such information for free (Fig. 15). As an additional improvement, since the dispatcher knows where its team members are, we have incorporated a very robust real-time monitoring system developed for past robotic projects that allows the dispatcher to put jobs back in the queue if a robot is not heard from for an extended period of time [45].

6 Conclusions

We have presented a distributed approach to mobile robot mapping and exploration. The system enables teams of robots to efficiently explore environments from

different, unknown locations. The robots initially explore on their own, until they can communicate with other robots. Once they can exchange sensor information with other robots, they estimate their relative locations using an adapted particle filter. To estimate whether or not the partial maps of two robots overlap, the filter incorporates a hidden Markov model that predicts observations outside the explored area. The parameters of the model are learned from previously explored environments using a hierarchical Bayesian approach. During exploration, the robots update their predictive models based on observations in the new environment. Our experiments indicate that this approach supports map merging decisions significantly better than alternative techniques.

The estimation of relative positions is integrated seamlessly into a decision-theoretic multirobot coordinated mapping strategy. To overcome the risk of false-positive map matches, the robots actively verify location hypotheses using a rendezvous strategy. If the robots meet at a meeting point, are able to ascertain their relative locations and can combine their data into a shared map. Mapping and map merging uses a SLAM technique that models uncertainty by local probabilistic constraints between the locations of laser range scans. Shared maps are used to coordinate the robots and to estimate the location of other robots.

The task inference algorithm we have described identifies potential team commitments that collectively balance constraints such as reachability, sensor coverage, and communication access. We have described a dispatch algorithm for task distribution and management that assigns resources depending on either task density or replacement requirements stemming from failures or power shortages. Since the target deployment environments are expected to lack a supporting communication infrastructure, robots manage their own network and reason about the concomitant localization constraints necessary to maintain team communication.

Our mapping and exploration system was evaluated under very strict real-world conditions. Prior to the evaluation, the developer team was allowed to test its robots in only half of the test environment. During the evaluation runs, the robots were forced to rely on their own adhoc wireless network to exchange information. The robots successfully explored the environment in all four official evaluation runs. All maps generated during these runs were virtually identical, indicating the high accuracy and robustness of the system.

Acknowledgements We thank Doug Gage and DARPA for support under the Software for Distributed Robotics Program (Contract #NBCHC020073). We also thank the evaluation team, Erik Krotkov and Douglas Hackett, for extraordinary efforts in designing and running the evaluation experiments. Part of this research was funded by the NSF under CAREER grant number IIS-0093406.

References

1. Arkin, R.C.: Behavior-based Robotics. MIT, Cambridge (1998)
2. Barraquand, J., Langlois, B., Latombe, J.C.: Robot motion planning with many degrees of freedom and dynamic constraints. In: Miura, H., Arimoto, S. (eds.) Robotics Research, vol. 5, pp. 435–444. MIT, Cambridge (1990)
3. Burgard, W., Moors, M., Fox, D., Simmons, R., Thrun, S.: Collaborative multi-robot exploration. In: Proc. of the IEEE International Conference on Robotics & Automation (ICRA) (2000)
4. Burgard, W., Moors, M., Stachniss, C., Schneider, F.: Coordinated multi-robot exploration. IEEE Trans. Robot. **21**, 376–386 (2005)

5. Dedeoglu, G., Sukhatme, G.S.: Landmark-based matching algorithm for cooperative mapping by autonomous robots. In: Proc. of the 5th International Symposium on Distributed Autonomous Robotic Systems (DARS) (2000)
6. Dissanayake, M.W.M., Newman, P., Clark, S., Durrant-Whyte, H.F., Csorba, M.: A solution to the simultaneous localization and map building (SLAM) problem. IEEE Trans. Robot. Autom. **17**(3), 229–241 (2001)
7. Keith Edwards, W.: Core Jini. Prentice Hall, Englewood Cliffs (2001)
8. Eliazar, A., Parr, R.: DP-SLAM: fast, robust simultaneous localization and mapping without predetermined landmarks. In: Proc. of the International Joint Conference on Artificial Intelligence (IJCAI) (2003)
9. Fenwick, J.W., Newman, P.M., Leonard, J.J.: Cooperative concurrent mapping and localization. In: Proc. of the IEEE International Conference on Robotics & Automation (ICRA) (2002)
10. Fox, D.: Adapting the sample size in particle filters through KLD-sampling. Int. J. Robot. Res. (IJRR) **22**(12) (2003)
11. Fox, D., Burgard, W., Dellaert, F., Thrun, S.: Monte Carlo localization: efficient position estimation for mobile robots. In: Proc. of the National Conference on Artificial Intelligence (AAAI) (1999)
12. Fox, D., Ko, J., Konolige, K., Limketkai, B., Stewart, B.: Distributed multi-robot exploration and mapping. In: Proc. of the IEEE. Special Issue on Multi-Robot Systems (2006)
13. Fox, D., Ko, J., Konolige, K., Limketkai, B., Stewart, B.: Distributed multi-robot exploration and mapping. In: Proc. of the IEEE, vol. 94(7). Special Issue on Multirobot Systems (2006)
14. Fox, D., Ko, J., Konolige, K., Stewart, B.: A hierarchical Bayesian approach to mobile robot map structure learning. In: Dario, P., Chatila, R. (eds.) Robotics Research: the Eleventh International Symposium. Springer Tracts in Advanced Robotics (STAR). Springer, New York (2005)
15. Gerkey, B., Mataric, M.: Multi-robot task allocation: Analyzing the complexity and optimality of key architectures. In: Proc. of the IEEE International Conference on Robotics & Automation (ICRA) (2003)
16. Grosz, B.J., Kraus, S.: Collaborative plans for complex group action. Artif. Intell. **86**(1), 269–357 (1996)
17. Gutmann, J.S., Konolige, K.: Incremental mapping of large cyclic environments. In: Proc. of the IEEE International Symposium on Computational Intelligence in Robotics and Automation (CIRA) (1999)
18. Hähnel, D., Burgard, W., Fox, D., Thrun, S.: An efficient FastSLAM algorithm for generating maps of large-scale cyclic environments from raw laser range measurements. In: Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (2003)
19. Howard, A., Parker, L.E., Sukhatme, G.S.: The SDR experience: experiments with a large-scale heterogenous mobile robot team. In: Proc. of the International Symposium on Experimental Robotics (ISER) (2004)
20. Jensfelt, P., Wijk, O., Austin, D., Andersson, M.: Feature based condensation for mobile robot localization. In: Proc. of the IEEE International Conference on Robotics & Automation (ICRA) (2000)
21. Ko, J., Stewart, B., Fox, D., Konolige, K., Limketkai, B.: A practical, decision-theoretic approach to multi-robot mapping and exploration. In: Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (2003)
22. Koenig, S., Tovey, C., Halliburton, W.: Greedy mapping of terrain. In: Proc. of the IEEE International Conference on Robotics & Automation (ICRA) (2001)
23. Konolige, K.: Large-scale map making. In: Proc. of the National Conference on Artificial Intelligence (AAAI) (2004)
24. Konolige, K.: SLAM via variable reduction from constraint maps. In: Proc. of the IEEE International Conference on Robotics & Automation (ICRA) (2005)
25. Konolige, K., Fox, D., Ortiz, C., Agno, A., Eriksen, M., Limketkai, B., Ko, J., Morisset, B., Schulz, D., Stewart, B., Vincent, R.: Centibots: very large scale distributed robotic teams. In: Ang, M., Khatib, O. (eds.) Experimental Robotics: the 9th International Symposium, Springer Tracts in Advanced Robotics (STAR). Springer, New York (2005)
26. Konolige, K.: A gradient method for realtime robot control. In: Proceedings of IROS (2000)
27. Konolige, K., Myers, K., Ruspini, E., Saffiotti, A.: The saphira architecture: a design for autonomy. J. Exp. Theor. Artif. Intell. **9** (1996)
28. Lenser, S., Veloso, M.: Sensor resetting localization for poorly modelled mobile robots. In: Proc. of the IEEE International Conference on Robotics & Automation (ICRA) (2000)

29. Lu, F., Milios, E.: Globally consistent range scan alignment for environment mapping. Auton. Robots **4**, 333–349 (1997)
30. Montemerlo, M., Thrun, S., Koller, D., Wegbreit, B.: FastSLAM: a factored solution to the simultaneous localization and mapping problem. In: Proc. of the National Conference on Artificial Intelligence (AAAI) (2002)
31. Moorehead, S.: Autonomous Surface Exploration for Mobile Robots. PhD Thesis, Carnegie Mellon University (2001)
32. Murphy, K.: Bayesian map learning in dynamic environments. In: Advances in Neural Information Processing Systems (NIPS) (1999)
33. Nettleton, E., Thrun, S., Durrant-Whyte, H.: Decentralised SLAM with low-bandwith communications for teams of airborne vehicles. In: Proc. of the International Conference on Field and Service Robotics (2003)
34. Newman, P., Leonard, J.J.: Consistent convergent constant time SLAM. In: Proc. of the International Joint Conference on Artificial Intelligence (IJCAI) (2003)
35. Ogier, R.G., Templin, F.L., Lewis, M.G.: Topology dissemination based on reverse-path forwarding. IETF RFC 3684 (Experimental) (2004)
36. Paskin, M.A.: Thin junction tree filters for simultaneous localization and mapping. In: Proc. of the International Joint Conference on Artificial Intelligence (IJCAI) (2003)
37. Pynadath, D.V., Tambe, M.: Automated teamwork among heterogeneous software agents and humans. J. Auton. Agents Multi-Agent Syst. **7**, 71–100 (2003)
38. Saffiotti, A., Ruspini, E.H., Konolige, K.: Integrating reactivity and goal-directedness in a fuzzy controller. In: Proceedings of the 2nd Fuzzy-IEEE Conference (1993)
39. Simmons, R., Apfelbaum, D., Burgard, W., Fox, D., Moors, M., Thrun, S., Younes, H.: Coordination for multi-robot exploration and mapping. In: Proc. of the National Conference on Artificial Intelligence (AAAI) (2000)
40. Stewart, B., Ko, J., Fox, D., Konolige, K.: The revisiting problem in mobile robot map building: a hierarchical Bayesian approach. In: Proc. of the Conference on Uncertainty in Artificial Intelligence (UAI) (2003)
41. Stroupe, A., Ravichandran, R., Balch, T.: Value-based action selection for exploration and dynamic target observation with robot teams. In: Proc. of the IEEE International Conference on Robotics & Automation (ICRA) (2004)
42. Thrun, S.: A probabilistic online mapping algorithm for teams of mobile robots. Int. J. Robot. Res. **20**(5) (2001)
43. Thrun, S.: Robotic mapping: a survey. In: Lakemeyer, G., Nebel, B. (eds.) Exploring Artificial Intelligence in the New Millenium. Morgan Kaufmann, San Francisco (2002)
44. Thrun, S., Burgard, W., Fox, D.: Probabilistic Robotics. MIT, Cambridge (2005)
45. Wilkins, D.E., Lee, T., Berry, P.: Interactive execution monitoring of agent teams. J. Artif. Intell. Res. **18**, 217–261 (2003)
46. Williams, S.B., Dissanayake, G., Durrant-Whyte, H.: Towards multi-vehicle simultaneous localisation and mapping. In: Proc. of the IEEE International Conference on Robotics & Automation (ICRA) (2002)
47. Yamauchi, B.: Frontier-based exploration using multiple robots. In: Proc. of the Second International Conference on Autonomous Agents (1998)
48. Zlot, R., Stentz, A., Bernardine Dias, M., Thayer, S.: Multi-robot exploration controlled by a market economy. In: Proc. of the IEEE International Conference on Robotics & Automation (ICRA) (2002)