Task Inference and Distributed Task Management in the Centibots Robotic System [*]

Charles L. Ortiz, Jr.
SRI International
333 Ravenswood Ave
Menlo Park, CA 94025
ortiz@ai.sri.com

Régis Vincent
SRI International
333 Ravenswood Ave
Menlo Park, CA 94025
vincent@ai.sri.com

Benoit Morisset
SRI International
333 Ravenswood Ave
Menlo Park, CA 94025
morisset@ai.sri.com

ABSTRACT

We describe the Centibots system, a very large scale distributed robotic system, consisting of more than 100 robots, that has been successfully deployed in large, unknown indoor environments, over extended periods of time (i.e., durations corresponding to several power cycles). Unlike most multiagent systems, the set of tasks about which teams must collaborate is not given *a priori*. We first describe a task inference algorithm that identifies potential team commitments that collectively balance constraints such as reachability, sensor coverage, and communication access. We then describe 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. The targeted 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. Finally, we present quantitative results in terms of a "search and rescue problem" and discuss the team-oriented aspects of the system in the context of prevailing theories of multiagent collaboration.

Categories and Subject Descriptors

I.2.9 [**Robotics**]: [Autonomous vehicles]; I.2.11 [**Distributed artificial intelligence**]: [Multiagent systems]

General Terms

Algorithms , Management, Performance, Experimentation

[*]This work is sponsored by the Defense Advanced Research Projects Agency (DARPA) Software for Distributed Robotic (SDR), under Department of the Interior Contract NBCHC020073. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright annotation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of DARPA, or the U.S. Government.

Keywords

Autonomous robots and robot teams, teamwork, mobile agents.

1. INTRODUCTION

Centibots, an autonomous and collaborative robotic system, has been successfully demonstrated on teams of more than 100 heterogeneous robots involved in joint exploration and searching tasks within indoor office-like environments. At deployment time, the area of operation for the robots is assumed to be unknown and also lacking any wireless communication infrastructure to support communication among robots. In addition, the Centibots system has been designed to operate over extended periods of time (i.e., durations corresponding to several power cycles).

From the robotics perspective, the motivation for studying such very large scale robotic systems is threefold: (1) the scale is such that tele-operation approaches would be infeasible; (2) multirobotic teams properly architected can provide a measure of survivability: if an element of the system is damaged or otherwise disabled, the system can continue to operate; and (3) division of labor means increased performance. In the second case, either assigning a separate operator per robot or requiring a single or small group of operators to monitor the *entire* team at once would result in, respectively, confusion among the operators or cognitive overload to an operator.

From the multiagent systems perspective, robotics has a number of appealing characteristics as a problem domain. In the first place, robotic applications can serve as testbeds in which important elements of multiagent theories become physically grounded (e.g., beliefs and actions) thereby leading to concrete measures of performance. Further, a multiagent theory is not as susceptible to the criticism that its success depends on unrealistic simplifying assumptions. For example, in robotics real-time performance is crucial and cannot be neglected. When developing Centibots, a number of challenges immediately arose. First, the task space over which the agents collaborate is "continuous" and not known in advance: the area of deployment could, in theory, be subdivided into an infinite number of subareas with a corresponding number of tasks. In addition, unlike traditional research in multiagent systems in which constraints on the communication medium are often abstracted away, in the Centibots system a robot cannot assume that when it comes time to communicate to another agent it can simply construct a suitable term in some agent communication language and then forward that to the receiving agent. The sending agent may need to reason about where to localize itself so that the message can be relayed through an intermediate agent; the agent must also be able to reason about the possibility of failure and and how to recover when necessary.

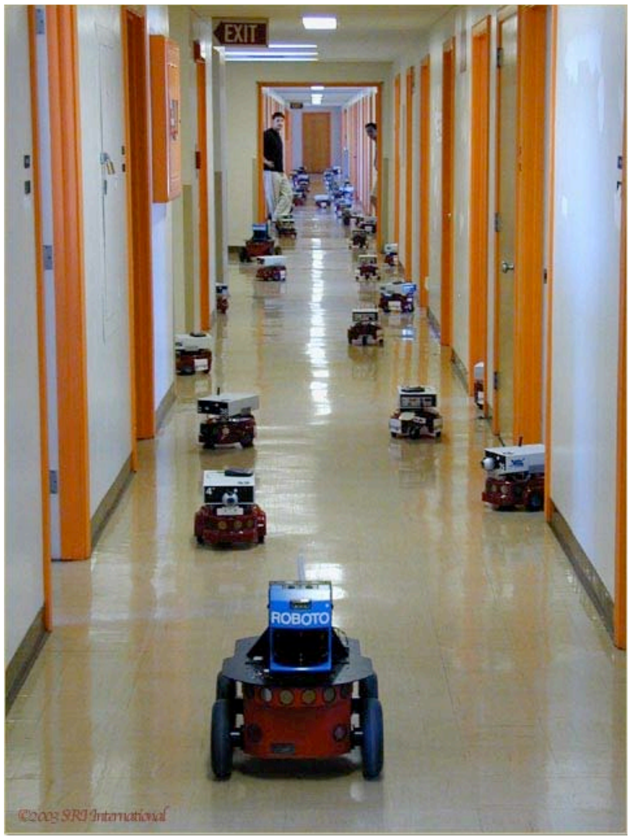

Figure 1: Centibots searching a 24,000 square foot unkown space

2. PROBLEM

The Centibots system has been tested in problem settings that correspond abstractly to "search and rescue" problems. These involve first finding an object of interest (OOI) within some large physical space and then "protecting" it by positioning robots in such a way as to maximize sensor coverage so that any intruders in the area can be tracked. Since the area in question is assumed to be unknown, such missions are articulated in three successive phases by the Centibots system: (1) a collection of *mapper* robots first conducts a coordinated exploration of the area and a concurrent mapping to produce an occupancy map of the environment; (2) the robots perform an exhaustive search for the OOI; and (3) the robots distribute themselves in a coordinated manner to cover the environment. In the real world, the OOI might correspond to an earthquake survivor or a hostage and the team of robots could be viewed as an initial response team deployed, for safety, in advance of a human crisis response team. Such robot teams can be viewed as extending a human's "eyes and ears" into some dangerous environment. The benefits of such a system are, therefore, potentially significant in terms of reduced risk to human life.

The robot fleet. The Centibots system includes search and mapper robots. The search team consists of 97 ActiveMedia Amigobot robots, each equipped with sonars, a wireless card, a processor consisting of a VIA Epia M9000 board (equivalent to a P3 933 processor), a monocular camera, and a 20 GB disk for storage. The mapping team consists of 6 ActiveMedia P2-A2 Pioneer robots with laser range finders, sonar, and onboard VersaLogic VSBC-8 boards with a Pentium III 850 MHz processor. Each robot can run localization, navigation, path planning and vision processing algorithms on its processor. For the experiments we will describe, the OOI was represented by a pink object (a ball or cube) which was easy to detect with onboard vision software. The form of the OOI was not important for the experiments; the purpose of the experiments was to demonstrate *collaborative search and exploration* and not perception.

Act types:

```
map(Group,Area)
search-for(Agent,Object,Area)
guard(Agent,Object)
replace(Agent1,Agent2)
video(Agent, Direction)
track(Agent,Object)
search(Agent,Room)
rotate(Agent,Angle)
```

Agent types: `mapper, explorer, link`
Agent states: `power, location, commitments, beliefs (e.g., map), task status`

Figure 2: Act and agent types in Centibots

In this paper, we summarize our research in mission stages (2) and (3) by way of a description of the SPARE (for SPAtial REasoning) system and then a distributed dispatcher algorithm used in the searching and protecting stages. Stage (1) was described in a previous paper [10].

3. TASK INFERENCE

Since none of the robots are equipped with effectors to manipulate their environment (e.g., for grasping objects), the set of *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. Examples are shown in Figure 2. Act types are represented as finite state machines in the Saphira programming language resident on each robot [12].

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. 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 the *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). The task nodes correspond to task (location) parameters for the task types shown in Figure 2. Once all of the parameters of an act type are instantiated, it becomes a fully instantiated act type (not yet necessarily executable): an example might be `search-for(robot32,OOI, N85)`. Once 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

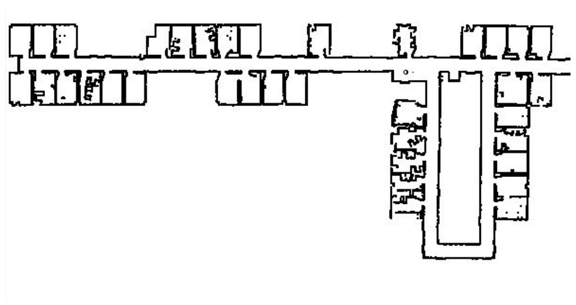
Figure 3: Occupancy map produced by first wave of mapper robots

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 [18]. As we will describe, the task inference process computes such behaviors by balancing several constraints.

3.1 Spatial representation in SPARE

The task inference process operates in a number of steps. First the occupancy map produced during the first mission stage is converted into *TG*, such that each vertex of *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 TG by first building a skeleton giving the structure of the free space and then generating 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 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 site, 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.

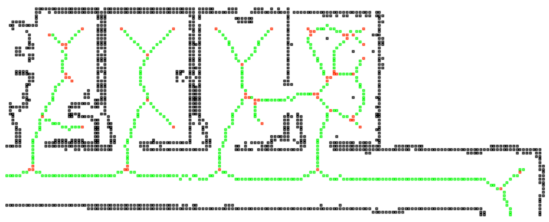
Figure 4: Part of the skeleton

4. Spatial information: for each edge, we associate the corresponding component of the Voronoi diagram

An additional step identifies the topological type 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

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 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", 1 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 alloca-

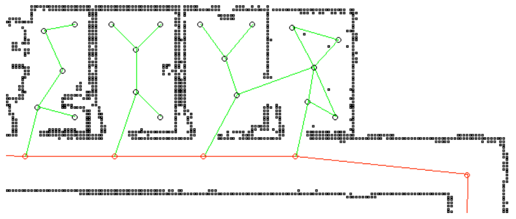

Figure 5: Part of the topological graph

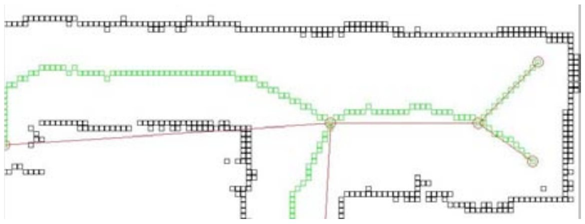

Figure 6: Skeleton components associated with edges

tions in corridors.

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 pre-computed 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.

3.2 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* (Figures 6 and 7).

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 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 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

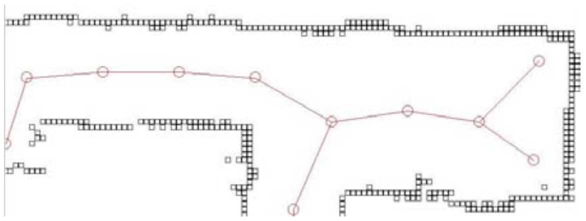

Figure 7: Addition of vertices along each component

solution was -1,540, with an average CPU time of 200 ms.

4. DISTRIBUTED TASK ALLOCATION

Having identified the tasks that need to be performed, the dispatching algorithm allocates robots to tasks and secures commitments for each robot. Recall that the Centibots system is intended to operate over multiple robot power cycles so that the dispatcher will also be responsible for reallocating resources. The system makes extensive use of the Jini [4] 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 [11]. 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 4.2 explains in more detail the differences in the two modes of operation.

4.1 Strategy and preference functions

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 difference that there is no *a priori* notion 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 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 24,000 square feet of

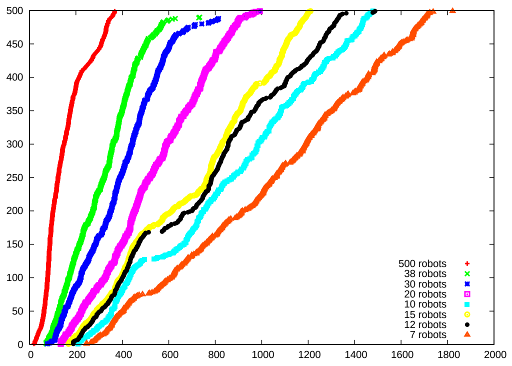

Figure 10: Increasing the number of robots available for searching beyond 20 does not yield a faster completion time

space.[1] The SPARE algorithm produced a TG with 499 nodes.

To simulate the environment in where 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 use 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* from one another and are not affecting each others. 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 performances, in the same setup, among alternative strategies. Figure 8 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 was worse than the graph predicted: this was due to 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 8 plots the performance of all strategies.

In contrast to the results shown in Figure 8, which were done in simulation, Figure 9 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.

4.2 Optimizations

The physical experiments identified the need for several enhancements to the clustering strategy. The first needed optimization had to do with the ad-hoc network employed by the system. [2] The

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

[2] We used the Topology Dissemination Based on Reverse-Path For-

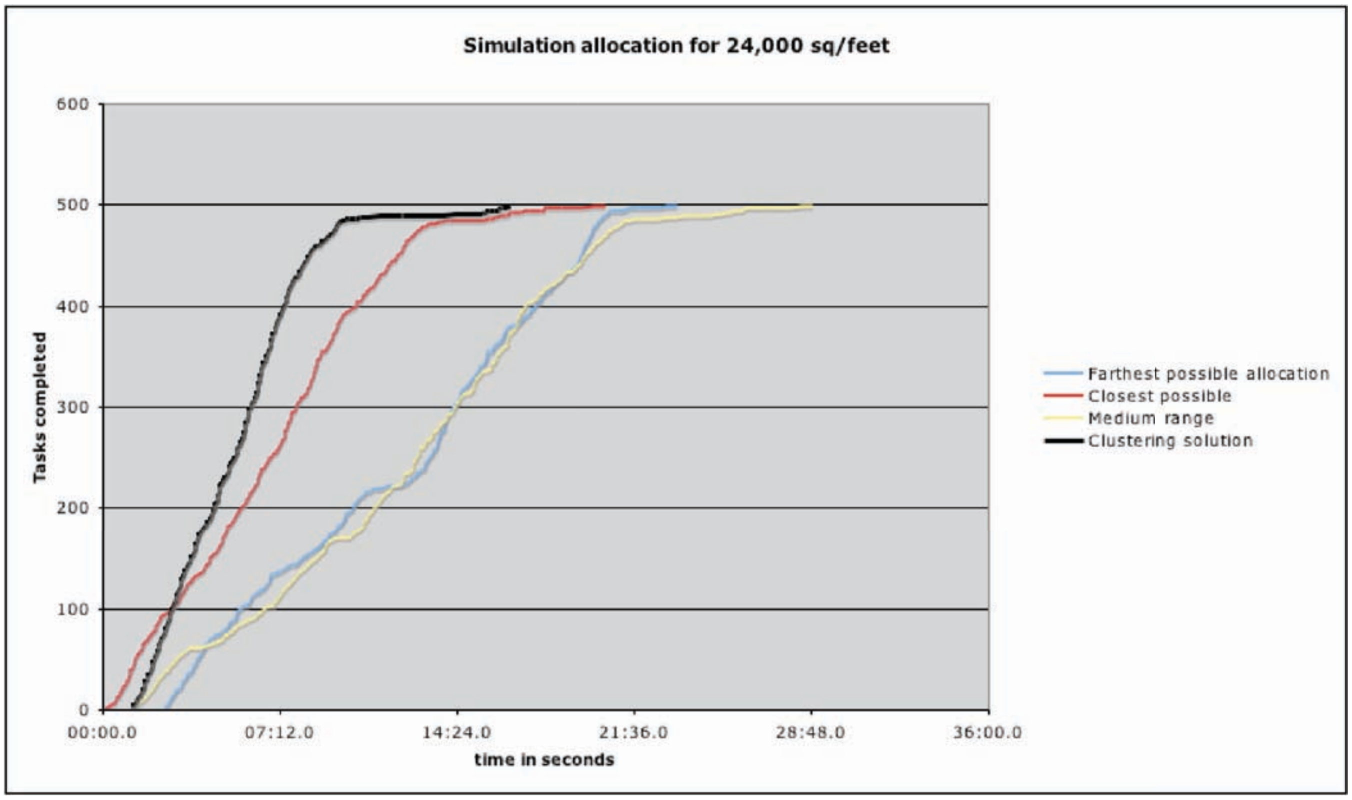

Figure 8: Comparison of all obvious strategies for task allocation. The X axis represents time and the Y axis represents the number of jobs completed. This data were collected in simulation.

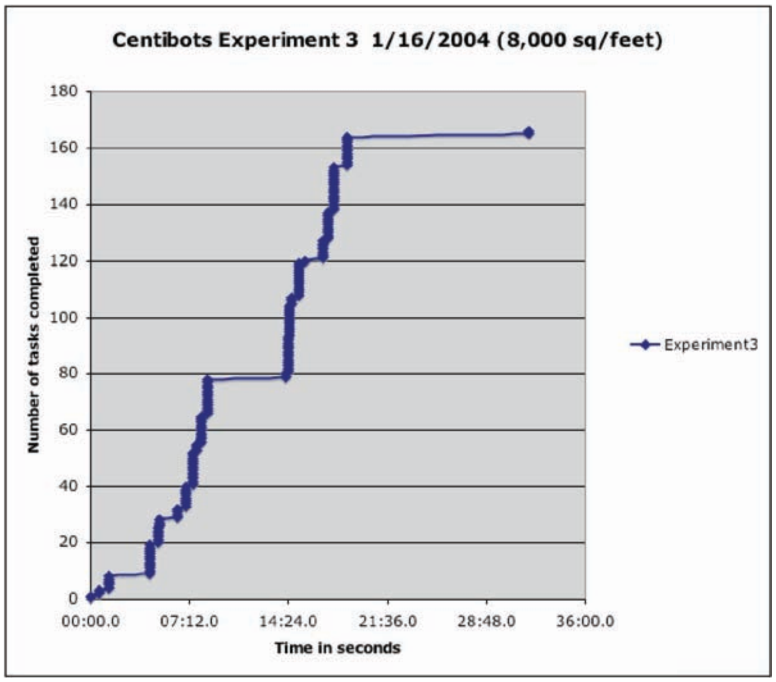

Figure 9: Graph of task allocation using the clustering method captured during an official experiment using 45 robots.The X axis represents time and the Y axis represents the number of jobs completed.

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 minutes 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 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 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.

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. 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 this 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. 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 [21].

warding (TBRPF) protocol [14].

5. 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 [5]. Within a real-time robotic system, explicitly representing and reasoning with such complex theories is a major challenge. Given the limited computational resources available to the sorts of simple robots that make up the Centibots system, it is more reasonable to use such theories as blueprints in the design; such an approach is one of those advocated in [5], 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 elements summarize the major requirements for successful collaboration over 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 [21], 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 [17] 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.

6. RELATED WORK AND LESSONS LEARNED

We have described the first successful, very large scale (on the order of 100 robots) distributed robotic system designed using multiagent principles (the other effort jointly funded by DARPA under this program instead adopted strictly emergent behavior approaches based on the notion of stigmergy [7,9]). Our approach makes use of decision theoretic techniques for task inference and auction-based approaches for distributed resource allocation. Our hope is that such approaches are more easily extendable when more complex behaviors than those found in social insects are desired. Although our approach did not make use of an explicit BDI architecture, we showed how the system satisfied many properties that have been ascribed to truly collaborative systems.

We introduced the distributed dispatcher idea in previous work: the Distributed Dispatcher Manager (DDM) [22] modeled thousands of agents and tasks and employed an algorithm similar to ours for resource allocation. There are a number of important differences, however: (1) tasks in DDM are given, corresponding to targets that appear in the environment and must be tracked; (2) the space through which sensor tracking agents move is unconstrained; and (3) the DDM experiments were restricted to simulation. Other work in distributed sensor networks, both mobile and stationary, are described in [13]. Target tracking with small teams of multiple robots in simulation have been examined in [8]. Deployment and exploration of a single mobile robot within a fixed communications network has been reported in [3]. Incremental deployment algorithms for 4-robot teams, without addressing communications constraints and without the use of an initial map of the deployment area, have been explored in [6]. Our work in interleaving communications planning and mission planning were originally developed and tested on teams of smaller outdoor robots equipped with GPS [15, 16, 19, 20].

We believe that we have also shown that robotics is a good domain for exploring multiagent system ideas: a number of technology enablers (smaller and cheaper platforms, faster processors, better sensors, wireless communication and ad-hoc network protocols) have made this possible. Among the challenges and lessons learned, from the multiagent systems perspective, were the following: (1) one cannot always assume that the set of multiagent tasks will be given *a priori*; (2) reasoning about communication and agent localization to support communication must be an integral part of agent systems; (3) real-time behavior in the robot domain is crucial; hence, complex deliberative reasoning must sometimes be substituted for faster reactive response; (4) the real world is unpredictable, demanding robustness of multiagent systems to action and resource failure; (5) agents will often need to satisfy multiple objectives at one time (e.g., search a room while acting as a communication link) and (6) operation in real space entails the vertical integration of many capabilities (such as sensor interpretation, localization and navigation).

7. ACKNOWLEDGMENTS

The authors would like to thank the entire Centibots team: Andrew Agno, Linda Briesemeister,Brian Burns, Michael Eriksen, Kurt Konolige , Mark Lewis, Enrique Ruspini, Dieter Fox, Jonathan Ko, Benson Limketkai, Bill Kennedy, Gary Fisher, Leonidas Guibas, Jean Claude LaTombe,Francois-Marie Lefevere,Danny Yang

8. REFERENCES

[1] R. C. Arkin. *Behavior-Based Robotics*. MIT Press, 1998.

[2] J. Barraquand, B. Langlois, and J.C. Latombe. Robot motion planning with many degrees of freedom and dynamic constraints. In H. Miura and S. Arimoto, editors, *Robotics Research*, volume 5, pages 435–444. MIT Press, 1990.

[3] Maxim A. Batalin and Gaurav S. Sukhatmen. Coverage, exploration, and deployment by a mobile robot and communication network. *Telecommunication Systems, Special Issue on Wireless Sensor Networks*, 26(2):181–196, 2004.

[4] W. Keith Edwards. *Core Jini*. Prentice Hall, 2001.

[5] Barbara J. Grosz and Sarit Kraus. Collaborative plans for complex group action. *Artificial Intelligence*, 86(1):269–357, 1996.

[6] Andrew Howard, Maja J. Mataric, and Gaurav Sukhatme. An incremental self-deployment algorithm for mobile sensor networks.

[7] Andrew Howard, Lynne E. Parker, and Gaurav S. Sukhatme. The sdr experience: Experiments with a large-scale heterogenous mobile robot team (extended abstract). In *9th International Symposium on Experimental Robotics 2004*, Singapore, Jun 2004.

[8] Boyoon Jung and Gaurav S. Sukhatme. Tracking targets using multiple robots: the effect of environment occlusion. *Autonomous Robots Journal*, 13(3):191–205, 2002.

[9] James Kennedy and Russell C. Eberhart. *Swarm Intelligence*. Academic Press, 2001.

[10] K. Ko, B. Stewart, D. Fox, K. Konolige, and B. Limketkai. A practical, decision-theoretic approach to multi-robot mapping and exploration. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2003.

[11] Kurt Konolige. A gradient method for realtime robot control. In *Proceedings of IROS*, 2000.

[12] Kurt Konolige, Karen Myers, Enrique Ruspini, and Alessandro Saffiotti. The saphira architecture: A design for autonomy. *Journal of Experimental and Theoretical AI*, 1996.

[13] V. Lesser, C. Ortiz, and M. Tambe, editors. *Distributed Sensor Networks: a multiagent perspective*. Kluwer Publishing, 2003.

[14] R.G. Ogier, F.L. Templin, and M.G. Lewis. Topology dissemination based on reverse-path forwarding, February 2004. IETF RFC 3684 (Experimental).

[15] Charles L. Ortiz and Eric Hsu. Structured negotiation. In *First International Conference on Autonomous agents and multiagent systems*, 2002.

[16] C.L. Ortiz, A. Agno, P. Berry, and R. Vincent. Multilevel adaptation in teams of unmanned air and ground vehicles. In *Proceedings of the First AIAA Unmanned Aerospace Vehicles, Systems, Technologies and Operations Conference and Workshop*, 2002.

[17] D.V. Pynadath and M. Tambe. Automated teamwork among heterogeneous software agents and humans. *Journal of Autonomous Agents and Multi-Agent Systems*, 7:71–100, 2003.

[18] A. Saffiotti, E. H. Ruspini, and K. Konolige. Integrating reactivity and goal-directedness in a fuzzy controller. In *Proceedings of the 2nd Fuzzy-IEEE Conference*, 1993.

[19] A. Saffiotti, N.B. Zumel, and E.H. Ruspini. Multirobot team coordination using desirabilities. In *In Proc. of the 6th Int. Conf on Intelligent Autonomous Systems (IAS)*, July 2000.

[20] R. Vincent, P. Berry, A. Agno, C. Ortiz, and D. Wilkins. Teambotica: a robotic framework for integrated teaming, tasking, networking, and control. In *Autonomous Agents and Multiagent Systems Conference*, 2003.

[21] D. E. Wilkins, T. Lee, and P. Berry. Interactive execution monitoring of agent teams. *Journal of Artificial Intelligence Research*, 18:217–261, March 2003.

[22] O. Yadgar, S. Kraus, and C. Ortiz. *Scaling up distributed sensor networks: cooperative large-scale mobile-agent organizations*, pages 185–218. Kluwer publishing, 2003.