Distributed Algorithms for Dispersion in Indoor Environments using a Swarm of Autonomous Mobile Robots

James McLurkin[1,2] and Jennifer Smith[1]

[1] iRobot Corporation,
Burlington, MA 01803
{jamesm, jsmith}@irobot.com
[2] Massachusetts Institute of Technology
Computer Science and Artificial Intelligence Laboratory
Cambridge, MA 02139
jamesm@csail.mit.edu

We describe a set of distributed algorithms used to disperse a large group of autonomous mobile robots efficiently throughout an indoor environment. Only local inter-robot communication and processing is used. Ad-hoc communications network topologies formed by gradient floods spread messages and guide robot motion. Special attention has been given to doors, hallways, and other constrictions. The network maintains a route to chargers to allow self-charging.

1 Introduction

Almost every application of swarms of robots requires them to disperse throughout their environment. Exploration, surveillance, and security applications all require coverage of large areas. In this work, we present algorithms for dispersing a large swarm of robots into an enclosed space. In order for a dispersion algorithm to work on physical robots, it must take into account engineering concerns – maintaining

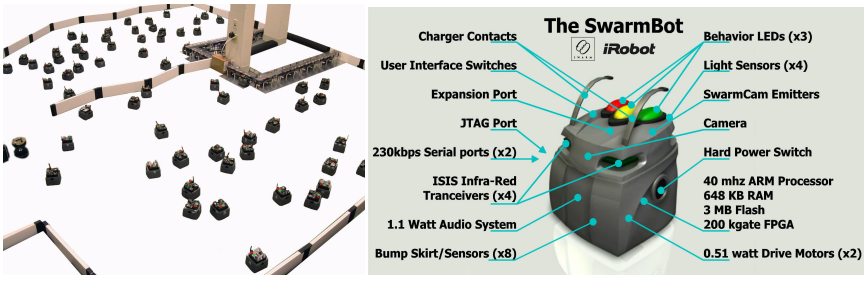


Fig. 1. The iRobot Swarm is comprised of over 100 SwarmBots™, 16 charging stations and navigational beacons. Each SwarmBot is roughly a 5" cube and has a suite of sensors, communications hardware, and human interface devices. Hands-free operation is important, thus the Swarm supports remote downloading and autonomous self-charging.

network connectivity, allowing for robot and communications failures, and providing an infrastructure for the robots to maintain their battery charge.

1.1 Swarm Hardware

The iRobot Swarm is shown in Fig. 1. Each SwarmBot™[1] contains an ARM Thumb CPU, a FPGA, a unique ID (UID) chip, a radio modem, serial ports, eight bump sensors, four light sensors, and a camera. Three extra-large LEDs and a 1.1 watt audio system provide user feedback.

The "Robot Ecology"™

Hands-free operation is critical, as even simple tasks, such as turning the robots on, become time consuming. The ″Robot Ecology″ infrastructure provides resources for centralized control, autonomous charging, and long-range navigation.

The ISIS™ Infrared Communication System

The primary sensor on the SwarmBot is an infrared inter-robot communication, location, and obstacle avoidance system called ISIS. Each robot has four ISIS transceivers, one in each corner. Nearby robots can communicate and determine the bearing, orientation, and range of their neighbors. The location system has a resolution of 1 cm and 2º at 30 cm range and a maximum range of 250 cm. A smaller range, $\mathbf{r_{safe}}$, is the maximum distance that provides reliable positioning. Reflected packets are used to determine the location of nearby obstacles and walls.

The Neighbor Cycle

Robots periodically transmit their externally visible state at the end of each neighbor cycle. This information includes their UID, what tasks they are performing, and any gradient messages they are relaying. We use the Aloha[1] protocol at the link layer. The period of the neighbor cycle, $\mathbf{t_n}$, is the same for all robots. This implies that each robot will receive only one communication from each of its neighbors during this period. These messages are collected and processed in a batch operation. This transforms the asynchronous distributed system into a synchronous distributed system, which greatly simplifies algorithm design. The period $\mathbf{t_n}$ is 250 ms, which allows for smooth robot motion control based on neighbor positions.

Behavior System

Swarm software is written as behaviors that run concurrently[2]. Each behavior returns a variable that contains actuator commands, i.e. motor velocities and light patterns.

Gradient Communication

A gradient-based multi-hop messaging protocol provides long-range communication using ISIS messages relayed from robot to robot. Gradients are used in many routing protocols to find optimal routes for messages through a network [3]. We use them to spread information and to guide robots through the network.

A source robot creates a gradient message that is relayed throughout the network in a breadth-first fashion, constructing a tree rooted at the source as it

[1] iRobot, ISIS, SwarmBot, Robot Ecology all copyright iRobot

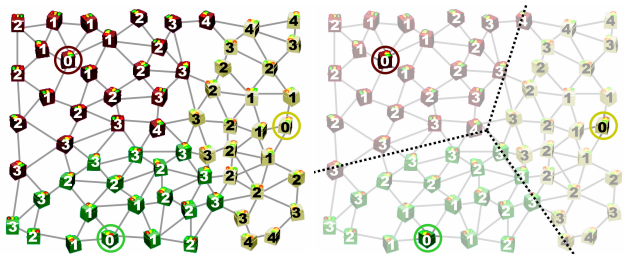


Fig. 2. Robots communicate with their neighbors over the gray lines in the left hand figure. Gradient-based routing protocols are used for long-range communication. Multiple gradient sources of the same type will tessellate the Swarm into Voronoi polygons, shown with the solid black lines in the figure on the right.

propagates. In each neighbor period, each robot processes the messages it has received and relays the one with the lowest hop count[2]. This eliminates cycles, and rebuilds the gradient tree each neighbor cycle to allow it to dynamically respond to changing network topologies as robots move.

With each robot transmitting periodically, but asynchronously, the expected time for any robot to receive a message during the neighbor period is

$$\mathbf{t_p} = \frac{\mathbf{t_n}}{2}.$$

The expected propagation time for a gradient message to disperse is

$$\mathbf{t_g} = \mathbf{diam(G)} \cdot \mathbf{t_p}$$

where $\mathbf{G}$ is the network graph, $\mathbf{diam(G)}$ is its diameter, and $\mathbf{t_n}$ is the neighbor cycle period. In the Swarm, the ideal value for $\mathbf{t_p}$ is 125 ms, but communications errors lengthen it to 172 ms. The diameter is dependent on the topography of the environment, but with 100 robots the practical limit is about 40 hops, resulting in a maximum propagation time of around 7 seconds. Multiple sources of the same gradient type will tessellate the Swarm as shown in Fig. 2.

Each robot retransmits messages every neighbor cycle. The total number of messages sent in an execution is given by:

$$\mathbf{n_m} = \left(\mathbf{c_{min\ msgs}} + \mathbf{n_g}\right)\frac{\mathbf{t}}{\mathbf{t_n}}\mathbf{n}$$

where $\mathbf{t}$ is the total running time of the algorithm, $\mathbf{n}$ is the total number of robots, $\mathbf{n_g}$ is the number of types of gradient messages, and $\mathbf{c_{min\ msgs}}$ is the minimum number of messages sent by a single robot, currently 4. Minimizing the number of messages per cycle per robot is an important design goal.

---

[2] If there are multiple packets with the same hop count, then the UID of the source and finally the UID of the sender is used as a tiebreaker. This deterministic tie-breaking procedure helps robots select the same neighbors to consider over multiple neighbor cycles, which reduces dithering between multiple equivalent neighbors.

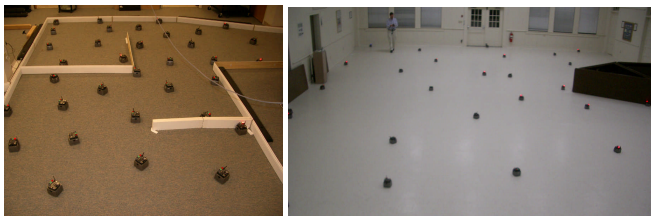


Fig. 3. **Left:** A dispersion into a small test space used to characterize the performance of different dispersion algorithms. This environment is approximately 6 m x 6 m, with several walls and "rooms". **Right:** A dispersion into a large room, note the person in the upper-left corner. It took about 20 minutes for them to achieve this dispersion, but they had to travel through a narrow hallway to get to this space, slowing their progress.

2 Directed Dispersion

The goal of the Directed Dispersion algorithm is to spread robots throughout an enclosed space quickly and uniformly, while keeping each robot connected to the network. The optimal running time occurs when each robot moves from a central start location to its final position along the shortest possible path at maximum velocity. Letting $\mathbf{E}$ be the closed polygon representing the environment, the minimum time for a dispersion is given by:

$$\mathbf{t_{min}} = \frac{\mathbf{diam(E)}}{\mathbf{v_{max}}}$$

where $\mathbf{diam(E)}$ is the maximum of shortest paths between any two points in the environment $\mathbf{E}$ and $\mathbf{v_{max}}$ is the maximum velocity of the robots. This minimum time is used to normalize the results of different algorithms.

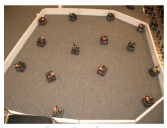
The dispersion is accomplished by using two algorithms that alternate running on the swarm: `disperseUniformly` and `frontierGuidedDispersion`. The `disperseUniformly` algorithm spreads robots evenly, using boundary conditions to limit the dispersion. The `frontierGuidedDispersion` algorithm directs robots towards unexplored areas, and is designed to perform well both in open environments and in environments with constrictions.

2.1 Uniform Dispersion - The `disperseUniformly` Algorithm

The `disperseUniformly` algorithm disperses robots uniformly throughout their environment. A thorough treatment of this technique is presented in [4]. Physical walls and a maximum dispersion distance between any two robots of $\mathbf{r_{safe}}$ are used as boundary conditions to help prevent the Swarm from spreading too thin and fracturing into multiple disconnected components.

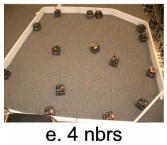
The algorithm works by moving each robot, $\mathbf{robot_i}$, away from the vector sum of the positions $\mathbf{p} = \{\mathbf{p}_1, \ldots, \mathbf{p_c}\}$ of their $\mathbf{c}$ closest neighbors $\mathbf{nbr} = \{\mathbf{neighbor}_1, \ldots, \mathbf{neighbor_c}\}$. The magnitude of the velocity vector that is given to the motor controller is:

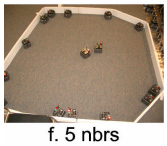

a. 0 nbrs

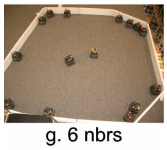

b. 1 nbrs

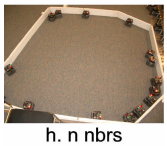

c. 2 nbrs


d. 3 nbrs


e. 4 nbrs


f. 5 nbrs


g. 6 nbrs


h. n nbrs

$$\mathbf{v} = \begin{cases} -\dfrac{\mathbf{v_{max}}}{\mathbf{c} \cdot \mathbf{r_{safe}}} \sum_{\mathbf{i}=1}^{\mathbf{c}} \mathbf{p_i} & |\mathbf{p_i}| \le \mathbf{r_{safe}} \\ 0 & |\mathbf{p_i}| > \mathbf{r_{safe}} \end{cases}$$

where $\mathbf{v_{max}}$ is the maximum allowable velocity output by this behavior. This vector directs the active robot away from its c nearest neighbors. The drive velocities are:

$$\mathbf{v_{rot}} = \mathbf{v} \cdot \mathbf{cos}(\mathbf{nbr_i.bearing}), \quad \mathbf{v_{trans}} = \mathbf{v} \cdot \mathbf{sin}(\mathbf{nbr_i.bearing})$$

where, $\mathbf{nbr_i.bearing}$, $\mathbf{nbr_i.range}$ is the bearing and range to $\mathbf{nbr_i}$.

This is a relaxation algorithm; imagine replacing graph $\mathbf{G}$ with its Delaunay triangulation $\mathbf{G'}$, and then placing compressed springs between connected robots. This will tend to expand the Swarm to fill the available space, but once the space is occupied, robots will position themselves to minimize the energy in the springs. Total group energy is minimized by minimizing local contributions, which happens when all the inter-robot distances are roughly equal. Fig. 4c and Fig. 3 show the robots uniformly dispersed in variously sized spaces.

The neighbors in $\mathbf{G'}$ are also Voronoi neighbors, the neighbors of the adjoining Voronoi cells of $\mathbf{robot_i}$. However, the robots are able to communicate across Voronoi cells, so the graph $\mathbf{G}$ usually has many edges that are not in the triangulation. This means that ISIS neighbors of a robot are not always Voronoi neighbors. Determining the set of Voronoi neighbors $\mathbf{nbr}$ from the set of ISIS neighbors, $\mathbf{nbr_{ISIS}}$, in real-time, is computation-intensive,[5] so an approximation is used. The closest ISIS neighbor will always be in the set $\mathbf{nbr}$. However, avoiding a single robot results in hectic movement as sensor errors can cause the position of this neighbor to change radically. Adding successively further neighbors to the set $\mathbf{nbr}$ cancels some of the position errors, but can also include non-Voronoi neighbors and cause the dispersion errors shown in Fig. 4a-h. When all elements of $\mathbf{nbr_{ISIS}}$ are added to $\mathbf{nbr}$, the robots are forced against the walls because the forces from distant neighbors from the other side of the circle are unbalanced.

In practice, using the two closest neighbors worked the best.

Fig. 4. The `disperseUniformly` algorithm is designed to spread the robots evenly. Instead of computing the closest neighbors (the neighbors of adjoining Voronoi polygons) to determine which robots to avoid, it avoids the n closest neighbors, sorted by range. Figs. a-h show the results of avoiding an increasing number of neighbors, with h showing the limit. Avoiding the two closest neighbors worked best in practice.

There are some cases in which second-closest neighbor is not a Voronoi neighbor, caused when the farther neighbor is "shadowed" by the closer neighbor. This case causes the robot to move in the same direction it would if only avoiding one neighbor, which does not cause errors, but does increase jitter. This "shadowing" effect is usually short lived, as the robot will typically encounter another neighbor or obstacle quickly.

2.2 Exploring New Areas - frontierGuidedDispersion

The goal of frontierGuidedDispersion is to guide robots towards areas they have yet to explore. Practical considerations require that the Swarm cannot fracture into disconnected components, as there must always be a route back to the chargers. The algorithm must self-stabilize to equalize voids and concentrations as robots enter and leave the network to charge. We also desire a termination condition to know when the Swarm is fully dispersed.

The frontierGuidedDispersion algorithm uses robots that are on the frontiers of explored space to guide the Swarm into unoccupied areas, similar to [6], but with support for multiple frontiers. The efficiency goal can be achieved if all the frontier robots move along their optimal path, leading the rest of the Swarm into their final positions.

Frontier Determination

Robots identify themselves as occupying one of three positions in the network: wall, frontier, or interior. "Wall" robots are those that detect an obstacle with the ISIS system. "Frontier" robots are those that have no neighbors and no walls within a large angle on any side; i.e., they are on the edge of an open space. The remainder are "interior" robots, as illustrated in Fig. 6. However, tight hallways require robots to become frontiers even when they detect walls. Fig. 6 shows how including the wall in the calculation of unoccupied space can correct this problem.

```
frontierDetermination() returns integer
1.  edgeNbrSet ⇐ set-of-all-neighbors
2.  if ISISWallSignalStrength > VirtualNeighborWallThreshold
3.     edgeNbrSet ⇐ edgeNbrSet ∪ createVirtualNbr(ISISRadar.bearing)
4.  endif

5.  edgeNbrSet ⇐ sortNbrsByBearing(edgeNbrSet )
6.  maxAngle ⇐ edgeNbrSet[1] + (360 - edgeNbrSet[length(edgeNbrs)] )
7.  for i ⇐ 2 to length(edgeNbrSet) − 1
8.     a ⇐ edgeNbrSet[i] - edgeNbrSet[i - 1]
9.     if a  > maxAngle
10.       maxAngle ⇐ a
11.    endif
12. endfor

13. if maxAngle > EdgeAngle
14.    return FrontierRobot
15. else if radar.range < WallRange
16.    return WallRobot
17. else
18.    return InteriorRobot
19. endif
```

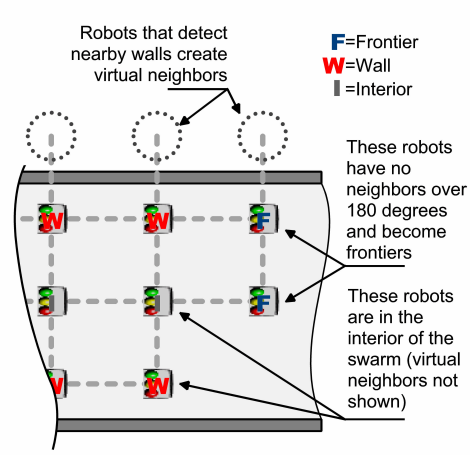


Fig. 5: Robots select their job as either frontier, wall, or interior robot based on the positions of their neighbors and nearby walls.

Lines 1-4 create a virtual neighbor if the global system variable ISISRadarSignal is greater than the VIRTUALNEIGHBORWALLTHRESHOLD. Lines 5-12 find the largest angle between any two adjacent robots. It does so by sorting the robots by bearing, then computing the difference in angle between adjacent elements. Lines 13-19 return the appropriate constant indicating the robot's position.

Swarm Motion - `disperseFromLeaves`

Once the robots know their positions in the network, the frontier robots source a gradient message. The trees created by these gradients ("frontier trees") guide the Swarm towards the frontier robots. It is possible to let the frontier robots "pull" the rest of the Swarm behind them by having the swarm cluster onto the frontier gradient source. However, any algorithm that is based on pulling robots over multiple hops can cause newly discovered frontiers to pull robots away from previously explored areas. This causes a frontier to re-appear at the old location and pull the Swarm back, creating oscillations, or fracturing the swarm and disconnecting robots from the chargers.

Instead, robots move away from children in the frontier tree. In order to build a reliable network, robots only move if they are in contact with at least two children in the frontier tree. This increases the min-cut of the network to two while the robots are dispersing, and helps deal with voids created by corners or robots heading home to charge.

`disperseFromLeaves(beh)`

```
1. childNbrSet ⇐ all-children-on-frontier-gradient-tree-closer-than-RSAFE
2. siblingNbrSet ⇐ all-siblings-on-frontier-gradient-tree-closer-than-RSAFE
3. if size(childNbrSet ) > 2
4.    avoidManyRobots(beh, (childNbrSet ∪ siblingNbrSet), d)
5. endif
```

Lines 1-2 create sets of children and sibling neighbors on the frontier tree that are closer than $\mathbf{r_{safe}}$. This limits the maximum dispersion to $\mathbf{r_{safe}}$. Line 3 requires a min-cut of 2 between this robot and its children. The `avoidManyRobots` behavior in line 4 takes the vector sum of the positions of the input set, and moves the robot in the opposite direction, i.e away from both sets of neighbors.

Leaves of the frontier tree remain stationary, which leaves robots in place to provide a route to the chargers and to mark previously explored areas. Essentially, the leaves become "anchors" and then limit the dispersion of robots away from

them to a distance of $\mathbf{r_{safe}}$.[3] As robots move away from the leaves, they move closer to their upstream robots, causing a chain reaction that eventually moves all the robots towards the frontiers.

Multiple frontiers often form as the Swarm explores the environment. Their gradients tessellate the Swarm based on hop count as shown in Fig. 2. This is useful because progress of distant frontiers will be slowed as interior robots disperse towards frontiers with smaller hop counts, allowing these closer frontiers to catch up. This tends to make the Swarm explore the building in a breadth-first fashion.

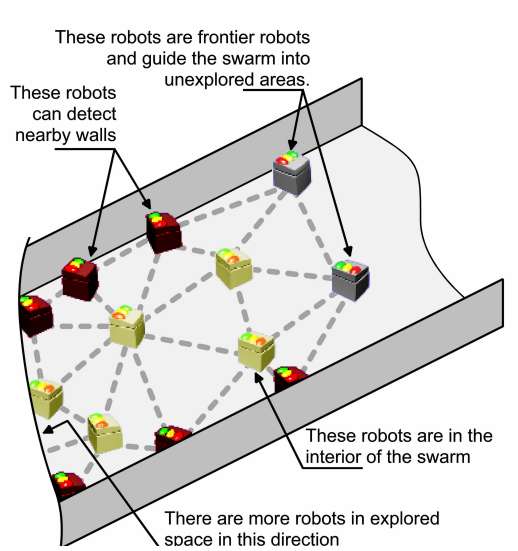


Fig. 6: Frontier robots guide the Swarm into unexplored areas by propagating a gradient that forms a tree rooted at the frontier robot. All robots then move away from their children in this tree. Leaves on the tree do not move, allowing previously dispersed robots to remain stationary.

2.3 Directed Dispersion

```
directedDispersion(beh)
1.  if frontierDetermination() = FrontierRobot
2.     gradientSource(FrontierGradient)
3.  endif

4.  if FrontierGradient.isActive = True
5.     disperseFromSource(beh)
6.  else
7.     disperseUniformly(beh)
8.  endif
```

Lines 1-3 source a FrontierGradient if this robot is on a frontier. Line 4 checks to see if there are any frontier gradients in the network, including the one from this robot. If so, line 5 runs the `disperseFromLeaves` behavior. Otherwise, `disperseUniformly` runs and equalizes inter-robot spacing. The "pressure" from `disperseUniformly` tends to push robots into open spaces and tight constrictions, which can cause new frontiers to form. This activates the `disperseFromLeaves` behavior on the rest of the swarm, which causes a directed dispersion towards the frontiers. The `disperseFromLeaves` behavior stays active until all frontiers encounter walls or move to the interior of the swarm. Termination of the combined

[3] Another way to think about this is to imagine that any robot that is not maximally dispersed from its children will head towards the frontier, causing its parent to move towards the frontier, etc. This results in a "wave" of motion that the frontier "surfs" forward .

algorithm is defined when the frontier behavior stays inactive for a specified amount of time. Unfortunately, complex environments, sensor noise, and robots leaving to charge can make it difficult to quantify this time. We used ten seconds for the experimental results.

3 Experimental Results

Experiments were conducted in February 2004, at iRobot in Burlington, Massachusetts. Fifty-six robots were used with a reduced ISIS communications power setting to explore the small environment shown on the left side of Fig. 3. There were three goals placed at varying distances from the start location. The Swarm was released and times required to reach the three goals and full dispersion were recorded. Five algorithms were compared.

`idealGasMotion`: Robots move in straight lines but turn when they collide with each other or with a wall. The network often breaks into disconnected components. Inter-robot interference is a problem, with robots colliding often. There is no termination condition, and dispersion is rarely uniform.

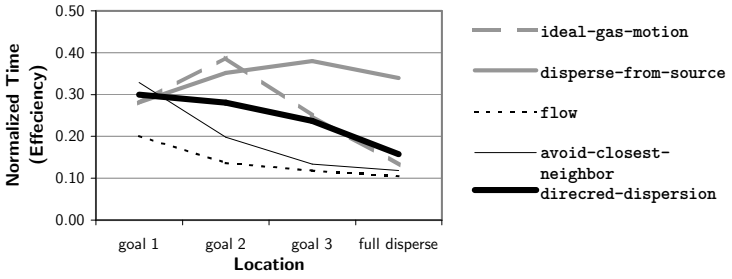
`disperseFromSource`: A robot near the base station sources a "disperse" gradient. Robots move a distance $\mathbf{r_{disperse}}$ away from parents in the "disperse" tree. Network connectivity is maintained during the dispersion process if $\mathbf{r_{disperse}} \leq \mathbf{r_{safe}}$. Uniform, complete coverage only occurs if the environment area is known in advance and $\mathbf{r_{disperse}}$ is set accordingly; otherwise robots either bunch up at boundaries or do not completely fill the area. However, the dispersion is very efficient, quickly reaching all goals and full dispersion.

`avoidClosestNeighbor`: Robots move away from their closest neighbor at constant velocity if $\mathbf{r} < \mathbf{r_{disperse}}$. Network connectivity can be maintained if $\mathbf{r_{disperse}} \leq \mathbf{r_{safe}}$. There is no termination condition. This is very similar to `disperseUniformly`, and the results are also similar. Dispersion is uniform, but robots oscillate back and forth between closest neighbors.

`disperseUniformly`: As described above in section 2.1. This algorithm runs more slowly than `avoidClosestNeighbor`, but the motion is smoother. It has very uniform dispersion and maintains network connectivity. Robots remain stationary after dispersion.

`directedDispersion`: As described in section 2.3. The robots rarely head in the

Table 1. Dispersion Efficiency vs. Location



wrong direction, and effectively push frontiers to the boundaries. The algorithm terminates with uniform coverage and robots remain stationary after dispersion.

Additional tests were conducted at a government-run experiment in a empty military schoolhouse in January 2004. A swarm of 108 robots dispersed into 3000 ft^2 of indoor space in about 25 minutes, located an object of interest, and led a human to it. Multiple room configurations were tested. The robots ran almost continuously for six hours, demonstrating the value of a number of features of the iRobot Swarm system: single-command activation, single-command return to base, fully integrated automatic recharging behavior, and the ability to ″bulk-reprogram″ the robots in the field.

4 Conclusion

Directed dispersion allows robots to explore large, complex, indoor environments. The robots use the information in the graph in which they are embedded to modify this same structure. Path planning and directed motion algorithms become easier to develop when the primary input is the positions of other nearby robots. Practical dispersion algorithms can be designed to meet efficiency, robustness, scalability, and correctness constraints.

Acknowledgments

This work was supported by DARPA IPTO under contracts SPAWAR N66001-99-C-8513 and SMDC DASG60-02-C-0028.

References

1 N. Abramson. ″The Aloha System - Another Alternative for Computer Communications″. In Proc. Fall Joint Cornput. Conf., AFIPS Conf., page 37, 1970.

2 R. Brooks. ″A robust layered control system for a mobile robot″. In IEEE Journal of Robotics and Automation, RA-2, pp.14-23, 1986.

3 C. Intanagonwiwat, R. Govindan and D. Estrin. Directed diffusion: A scalable and robust communication paradigm for sensor networks. In Proc. Sixth Annual International Conference on Mobile Computing and Networks, 2000.

4 J. Cortes, S. Martinez, T. Karatas, and F. Bullo. Coverage control for mobile sensing networks. In Proceedings of the IEEE International Conference on Robotics and Automation, pages 1327--1332, Arlington, VA, May 2002.

5 S. Arya and A. Vigneron. ″Approximating a Voronoi Cell″. HKUST Theoretical Computer Science Center Research Report HKUST-TCSC-2003-10, Hong Kong University of Science and Techology, available at www.comp.nus.edu.sg/~antoine/avn.pdf, 2003.

6 D. Payton, M. Daily, R. Estowski, M. Howard, and C. Lee. ″Pheromone Robotics″. In Autonomous Robots, vol. 11, pp.319-324, 2001.