Event-Based Motion Control for Mobile-Sensor Networks

Many sensor networks have far more units than necessary for simple coverage. Sensor mobility allows better coverage in areas where events occur frequently. The distributed schemes presented here use minimal communication and computation to provide this capability.

Zack Butler and Daniela Rus
Dartmouth College

In many sensor networks, considerably more units are available than necessary for simple coverage of the space. Augmenting sensor networks with motion can exploit this surplus to enhance sensing while also improving the network's lifetime and reliability. When a major incident such as a fire or chemical spill occurs, several sensors can cluster around that incident. This ensures good coverage of the event and provides immediate redundancy in case of failure.

Another use of mobility comes about if the specific area of interest (within a larger area) is unknown during deployment. For example, if a network is deployed to monitor the migration of a herd of animals, the herd's exact path through an area will be unknown beforehand. But as the herd moves, the sensors could converge on it to get the maximum amount of data. In addition, the sensors could move such that they also maintain complete coverage of their environment while reacting to the events in that environment. In this way, at least one sensor still detects any events that occur in isolation, while several sensors more carefully observe dense clusters of events.

We've developed distributed algorithms for mobile-sensor networks to physically react to changes or events in their environment or in the network itself (see the "Related Work" sidebar for other approaches to this problem). Distribution supports scalability and robustness during sensing and communication failures. Because of these units' restricted nature, we'd also like to minimize the computation required and the power consumption; hence, we must limit communication and motion. We present two classes of motion-control algorithms that let sensors converge on arbitrary event distributions. These algorithms trade off the amount of required computation and memory with the accuracy of the sensor positions. Because of these algorithms' simplicity, they implicitly assume that the sensors have perfect positioning and navigation capability. However, we show how to relax these assumptions without substantially affecting system behavior. We also present three algorithms that let sensor networks maintain coverage of their environment. These algorithms work alongside either type of motion-control algorithm such that the sensors can follow the control law unless they must stop to ensure coverage. These three algorithms also represent a trade-off between communication, computation, and accuracy.

Controlling sensor location

We assume that events of interest take place at discrete points in space and time within a given area. If those events come from a particular distribution, which can be arbitrarily complex, the sensors should move such that their positions will eventually approximate that distribution. In addition, we'd like to minimize the amount of neces-

 Published by the IEEE CS and IEEE ComSoc ■ 1536-1268/03/$17.00

Related Work

Previous work in sensor networks has inspired this work. We've built on our own work on routing in ad hoc networks[1] and reactive sensor networks.[2] We've also built on important contributions from other groups.[3–5] Massively distributed sensor networks are becoming a reality, largely due to the availability of mote hardware.[6] Alberto Cerpa and Deborah Estrin propose an adaptive self-configuring sensor network topology in which sensors can choose whether to join the network on the basis of the network condition, the loss rate, the connectivity, and so on.[7] The sensors do not move, but the network's overall structure adapts by causing the sensors to activate or deactivate. Our work examines mobile-sensor control with the goal of using redundancy to improve sensing rather than optimize power consumption.

Researchers have only recently begun to study mobile-sensor networks. Gabriel Sibley, Mohammad Rahimi, and Gaurav Sukhatme describe the addition of motion to Mote sensors, creating *Robomotes*.[8] Algorithmic work focuses mainly on evenly dispersing sensors from a source point and redeploying them for network rebuilding,[9,10] rather than congregating them in areas of interest. Related work by Jorge Cortes and his colleagues[11] uses Voronoi methods to arrange mobile sensors in particular distributions, but in an analytic way that requires defining the distributions beforehand. Our work focuses on distributed reactive algorithms for convergence to unknown distributions—a task that researchers have not previously studied.

REFERENCES

1. Q. Li and D. Rus, "Sending Messages to Mobile Users in Disconnected Ad Hoc Wireless Networks," *Proc. 6th Ann. Int'l Conf. Mobile Computing and Networking* (MOBICOM 00), ACM Press, 2000, pp. 44–55.
2. Q. Li, M. DeRosa, and D. Rus, "Distributed Algorithms for Guiding Navigation across Sensor Networks," *Proc. 9th Ann. Int'l Conf. Mobile Computing and Networking* (MOBICOM 03), ACM Press, 2003, pp. 313–325.
3. G.J. Pottie, "Wireless Sensor Networks," *Proc. IEEE Information Theory Workshop*, IEEE Press, 1998, pp. 139–140.
4. J. Agre and L. Clare, "An Integrated Architecture for Cooperative Sensing Networks," *Computer*, vol. 33, no. 5, May 2000, pp. 106–108.
5. D. Estrin et al., "Next Century Challenges: Scalable Coordination in Sensor Networks," *Proc. 5th Ann. Int'l Conf. Mobile Computing and Networking* (MOBICOM 00), ACM Press, 1999, pp. 263–270.
6. J. Hill et al., "System Architecture Directions for Network Sensors," *Proc. 9th Int'l Conf. Architectural Support for Programming Languages and Operating Systems* (ASPLOS 00), ACM Press, 2000, pp. 93–104.
7. A. Cerpa and D. Estrin, "Ascent: Adaptive Self-Configuring Sensor Networks Topologies," *Proc. 21st Ann. Joint Conf. IEEE Computer and Communications Societies* (INFOCOM 02), IEEE Press, 2002, pp. 1278–1287.
8. G.T. Sibley, M.H. Rahimi, and G.S. Sukhatme, "Robomote: A Tiny Mobile Robot Platform for Large-Scale Sensor Networks," *Proc. IEEE Int'l Conf. Robotics and Automation* (ICRA 02), IEEE Press, 2002, pp. 1143–1148.
9. M.A. Batalin and G.S. Sukhatme, "Spreading Out: A Local Approach to Multi-robot Coverage," *Proc. Int'l Conf. Distributed Autonomous Robotic Systems 5* (DARS 02), Springer-Verlag, 2002, pp. 373–382.
10. A. Howard, M.J. Mataric, and G.S. Sukhatme, "Mobile Sensor Network Deployment Using Potential Fields: A Distributed, Scalable Solution to the Area Coverage Problem," *Proc. Int'l Conf. Distributed Autonomous Robotic Systems 5*, Springer-Verlag, 2002, pp. 299–308.
11. J. Cortes et al., "Coverage Control for Mobile Sensing Networks," *IEEE Int'l Conf. Robotics and Automation* (ICRA 03), IEEE Press, 2003, pp. 1327–1332.

sary computation, memory, and communication, while still developing distributed algorithms. Each sensor, therefore, must approximate the event distribution and must position itself correctly with respect to it. In particular, for scalability, we don't consider strategies where each sensor maintains either the entire event history or the locations of all other sensors. We assume that at least one sensor can sense each event and broadcast the event location to the other sensors, so that every sensor learns about each event location. (We don't consider the particular mechanism of this broadcast in this article.) If the initial distribution is uniform, either random or regular, then the sensors can move on the basis of the events without explicitly cooperating with their neighbors. The two motion-control algorithms we present here both use this observation, but they differ in the amount of storage they use to represent the history of sensed events.

History-free techniques

In this class of motion-control algorithms, the sensors don't maintain any event history. This approach resembles the potential–field approaches in formation control and coverage work,[1] which use other robots' current positions to determine motion. The main difference is that our approach considers event, rather than neighbor, positions. This technique is appealing due to its simple nature and minimal computational requirements. Here we allow each sensor to react to an event by moving according to a function of the form

$$x_i^{k+1} = x_i^k + f\left(e^{k+1}, x_i^k, x_i^0\right),$$

where e^k is the position of event k, and x_i^k refers to the position of sensor i after event k.

The form of function f in this equation is the important component of this strategy. For example, one simple candidate function,

$$c\left(e^{k+1} - x_i^k\right),$$

which treats positions as vector quantities, causes the sensor to walk toward the event a short distance proportional to how far it is from the event. Although

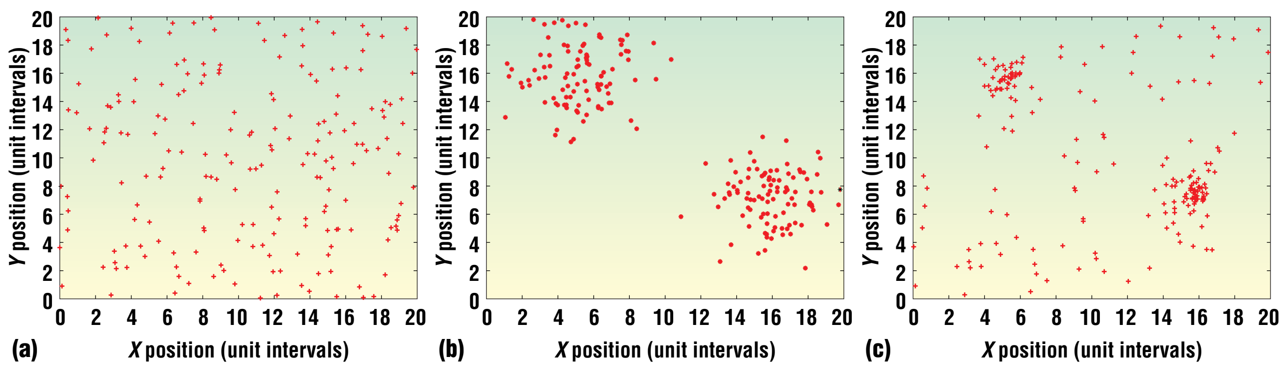


Figure 1. The results of a mobile-sensor simulation using a history-free update rule (with $\alpha = 0.06$, $\beta = 3$, $\gamma = 1$): (a) the initial sensor positions, generated at random; (b) the positions of a series of 200 events; and (c) the final sensor positions.

simple, this turns out not to be a good choice for most event distributions, because it causes all the sensors to cluster around the mean of all events. In fact, many such update functions have this effect.

We can identify several useful properties for f. First, after an event occurs, the sensor should never move past that event. Second, the sensors' motion should tend to 0 as the event gets further away, so that the sensors can separate themselves into multiple clusters when the events are likewise clustered. Finally, it's reasonable to expect the update to be monotonic; no sensor should move past another along the same vector in response to the same event.

One way to restrict the update function is to introduce a dependency on the distance d between the sensor and the event, and then always move the sensor directly toward the event. We can ensure the desired behavior, using these three criteria:

$$\forall\, d,\ 0 \le f(d) \le d$$
$$f(\infty) = 0$$
$$\forall\, d_1 > d_2,\ f(d_1) - f(d_2) < (d_1 - d_2)$$

One simple function that fulfills these criteria is $f(d) = de^{-d}$ (where e here refers to the constant 2.718…, not an event). We can also use other functions in the family $f(d) = \alpha d^{\beta} e^{-\gamma d}$ for values of parameters α, β, and γ such that $\alpha e^{-\gamma d}(\beta d^{\beta-1} - \gamma d^{\beta}) > 1\ \forall\, d$. We've implemented simulations using several functions in this family as update rules, and Figure 1 shows the results of using this technique (with $\alpha = 0.06$, $\beta = 3$, $\gamma = 1$). For this particular family of functions, the parameters can change over a wide range and still produce fairly reasonable results, differing in their convergence speed (primarily dependent on α) and in the region of influence of a cluster of events (dependent on β and γ).

History-based techniques

The preceding algorithm needs only minimal information. The resulting sensor placement is acceptable for many applications, but with a small amount of additional information, we can improve it. Here we explore the benefits of maintaining event history to improve the sensors' approximation of the event distribution. Sensors can use history at each update to make more informed decisions about where to go at each step. Letting them build a transformation of the underlying space into a space that matches the event distribution makes this possible. To limit the amount of necessary memory, this algorithm doesn't keep the location of every event. Instead, a coarse histogram over the space serves to fix memory use beforehand.

A one-dimensional algorithm. The simplest instantiation of this concept is in one dimension. 1D event distributions can enable mapping for many applications—for example, monitoring roads, pipelines, or other infrastructure. Here, the transformed space is simply a mapping using the events' *cumulative distribution function*.

To determine its correct position, each sensor maintains a discrete version of the CDF, which updates after each event. We scale the CDF on the basis of the number of events and length l of the particular interval, such that $CDF(l) = l$. We then associate each segment of the CDF with a proportional number of sensors so that the sensor density tracks the event density. Because the sensors are initially uniformly distributed, we can accomplish this by mapping each CDF segment to a proportional interval of the sensors' initial positions. Each sensor calculates its correct transformed position on the basis of the inverse of the CDF, evaluated at its initial position. In other words, a sensor chooses the new position such that the CDF at this position returns its initial position. The algorithm in Figure 2 describes this process.

This algorithm produces an approximately correct distribution of sensors because the number of sensors that map their current position into the original x–axis interval is proportional to the event density in that interval. In addition, the mapping ensures that a sensor that starts at a given fraction of the way along the interval moves so that it keeps the same fraction of events to its left. Moreover, because the CDF is monotonic, no sensor will pass another when reacting to an event.

for event at position e^k **do**
    Increment CDF bins representing positions $\geq e^k$
    Scale CDF by $k/(k+1)$
    Find bins b_i and b_{i+1} with values $b_i \leq x_0 \leq b_{i+1}$
    Compute position x_c by interpolation of values of b_i and b_{i+1}

Figure 2. A one-dimensional history–based algorithm.

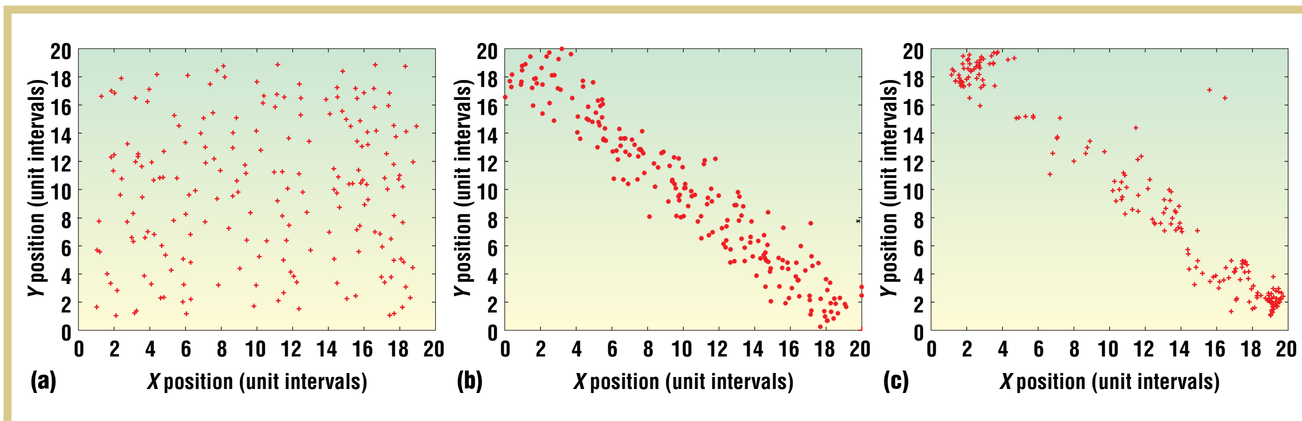
A two-dimensional algorithm. Although the 1D algorithm has some potential practical applications, many other monitoring applications over planar domains exist, such as monitoring forest fires. However, we can extend the 1D algorithm by building a 2D histogram of the events and using it to transform the space similarly. After each event, every sensor updates the transformed space on the basis of the event position and determines its new position by solving a set of 1D problems using the algorithm in Figure 2.

When an event occurs, each sensor updates its representation of the events. This is the same as incrementing the appropriate bin of an events histogram, although the sensors don't represent the histogram explicitly. Instead, each sensor keeps two sets of CDFs, one set for each axis. That is, for each row or column of the 2D histogram, the sensor maintains a CDF, scaled as in the 1D algorithm. We use this representation rather than a single 2D CDF, in which each bin would represent the number of events below and to the left, because this latter formulation would induce unwanted dependency between the axes. In a single 2D CDF, events occurring in two clusters, such as in Figure 1b, would induce a third cluster of sensors in the upper right.

After the sensor has updated its data structure, it searches for its correct next position. To do this, it performs a series of interpolations as in the 1D algorithm. For each CDF aligned with the x-axis, the sensor finds the value corresponding to its initial x-coordinate, and likewise for the y-axis. This creates two sets of points, which can be viewed as two chains of line segments: one going across the workspace (a function of x) and one that's a function of y. We can also view these chains as a constant height contour across the surface defined by the CDFs. To determine its next position, a sensor looks for a place where these two segment chains intersect. However, given the nature of these chains' construction, more than one such place is possible. So, our algorithm directs the sensor to go to the intersection closest to its current position. This is somewhat heuristic but is designed to limit the required amount of motion, and in practice it appears to produce good results. Figure 3 shows typical results, similar to those of other event distributions.

Because this algorithm updates only one bin of the histogram, the computation necessary for the CDF update is low, equivalent to two 1D calculations, and the time for the position calculation is proportional to the histogram width. In addition, the algorithm has the useful property that two sensors not initially collocated won't try to move to the same point. Finally, unlike the his-

Figure 3. Results of the history–based algorithm: (a) the initial sensor positions, generated randomly; (b) the positions of a series of 200 events; and (c) the sensors' final positions.



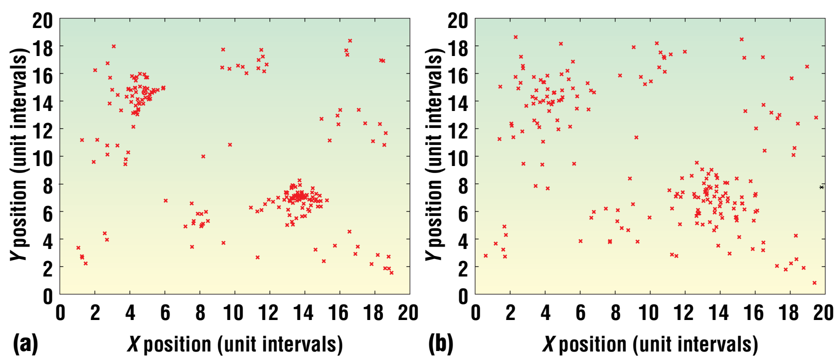


Figure 4. Results of a mobile-sensor simulation under the history-based algorithm: (a) the final positions of the sensors without noise and (b) the final positions of the sensors with noise of 25 percent deviation for each motion.

tory-free algorithms presented earlier, this technique will correctly produce a uniform distribution of sensors, given a uniform distribution of events, because each CDF will be linear, and the initial position's mapping to the current position will be the identity.

Handling uncertainty

The preceding algorithms implicitly assume that each sensor knows its current position and can move precisely to its desired position at any time. Here we briefly describe the effects of relaxing this assumption. Intuitively, we expect that because these approaches rely on many sensors in a distribution, nonsystematic errors will tend not to bias the resulting sensor distribution. For example, if each sensor has a small Gaussian error in its perceived initial position, the perceived initial distribution will still be close to uniformly random (in fact, it will be the convolution of the uniform distribution with the Gaussian). Similarly, if event sensing is subject to error, the sensors will converge toward a distribution that's the true error distribution convolved with the sensing error's distribution.

When the sensors move under our algorithms' control, the situation's complexity increases somewhat. If we envision each sensor as a Gaussian blob around its true position, and each motion of the sensor induces additional uncertainty, the sensor's true position will be a convolution of these two distributions. Over time, we would expect the resultant sensor distribution to be a smoothed version of the intended distribution. This applies equally to both the history-free and the history-based algorithms. Although the latter use only the initial position to compute the intended position, whereas the former use only the current position, the position error should accumulate in the same way (assuming each position is correct). One difference is that the history-based algorithm might involve more sensor motion and, therefore, more opportunity to accumulate error.

To examine this intuition empirically, we included noise models for initial-position and motion error in the Matlab simulations. Initial-position noise is Gaussian, whereas we model motion error as an added 2D Gaussian noise whose variance is proportional to the distance moved. Figure 4 shows typical results, with the same set of initial positions and events, running with and without noise.

Maintaining coverage of the environment

Now, we extend the event-driven control of sensor placement to include coverage of the environment. Under the algorithms thus far presented, sensor networks can lose network connectivity or sensor coverage of their environment. The ability to maintain this type of coverage while still reacting to events is an important practical constraint because it can guarantee that the network remains connected and monitors the entire space. This way, the network can still detect and respond to new events that appear in currently "quiet" areas.

We assume that each sensor has a limited communication and sensing range, and at least one sensor should sense every point in the environment. Every sensor moves to maintain coverage, or, if not required for coverage, follows the event distribution exactly. This is similar to space-filling coverage methods, such as those that use potential fields.[1] In these methods, each robot moves away from its colleagues to produce a regular pattern in the space and thereby complete coverage. You can extend these space-filling methods to the variable-distribution case by changing the potential field strengths on the basis of the event distribution. In our work, however, the sensors follow the event distribution exactly until required for coverage. They can thus achieve a good distribution approximation in high-density areas and good coverage in low-density areas. This switching technique also simplifies prediction of other sensors' motions.

Recall that in both the history-free and the history-based algorithms, each sensor moves according to a simple known control function. Each sensor can therefore predict the motion of other sensors and use this information to maintain adequate coverage. Prediction of other sensor positions requires additional computation, which can be significant if the update algorithm is complex or there are many sensors to track. We can avoid this computation by using communication whereby each sensor broadcasts its position to nearby sensors. However, more communication also has potential drawbacks in terms of power use.

Here we present three different methods for maintaining coverage that use different amounts of communication and computation, and we compare their performance. Each algorithm can work with either the history-free or the history-based motion-control algorithms.

Coverage using communication

The first algorithm we describe uses communication to ensure coverage. Under this protocol, each sensor maintains a circular area of interest around its current position, and attempts to keep that area spanned by other sensors. This implicitly assumes that each device's communication and sensing range is circular. Depending on the task, the area size can relate to either the device's communication range or sensor range. After each event, each sensor broadcasts its new position to its neighbors to aid coverage. Because this information is useful only to the sensors in the broadcasting sensor's neighborhood, this position message does not propagate; so, this scheme is scalable to large networks.

To ensure that coverage is complete, after each event, each sensor examines the locations of the sensors in its neighborhood. If any semicircle within its area of interest is empty, no neighbor covers a portion of that area. This indicates a potential loss of coverage. Figure 5 gives the algorithm for detecting empty semicircles, and Table 1 lists this algorithm's properties. Once the sensor has learned its neighbors' positions, it calculates the relative angle to each neighbor. The sensor then sorts these angles; any gap between neighbors equal to π or greater indicates an empty semicircle.

for each neighbor position (x_j, y_j) **do**
    $\theta_j = \arctan[(y_j - y)/(x_j - x)]$
Sort θ_j into vector $\Phi(\Phi_0 \ldots \Phi_n)$
$\Phi_{n+1} = \Phi_0 + 2\pi$
$\Delta\Phi_k = \Phi_{k-1} - \Phi_k$
if $\max_k(\Delta\Phi_k) > \pi$ **then**
    Empty area exists, handle as in text

Figure 5. A communication–based algorithm for ensuring coverage (where θ is a vector of angles to neighbors and Φ is a sorted vector of angles).

An empty semicircle within a sensor's area of interest indicates potential loss of coverage. When the sensor finds such an empty area, it must employ an appropriate strategy to ensure coverage. The first option is simply to remain fixed at its current position. The second option is to move a small distance toward the middle of the open semicircle. The distance should be small enough so that no other neighbors move outside the area. This latter option allows more even coverage but makes predicting other sensors' positions far more computationally expensive, so this option is incompatible with the predictive methods described next.

This reactive method for ensuring coverage is appealing because it requires little additional computation and is still scalable. However, it's limited because it considers only those sensors that are within its communication range, R_c.

Predictive methods

Now we describe a way to ensure coverage based on predicting other sensors' positions. This method involves constructing *Voronoi diagrams* to determine whether complete coverage exists. (A Voronoi diagram divides a plane into regions, each consisting of points closer to a given sensor than to any other sensor.) This approach reduces double coverage at the expense of the additional computation required to calculate the Voronoi diagram. We assume each sensor knows its initial position. In the algorithm's initialization phase, each sensor broadcasts this position, letting every other sensor track that sensor's location.

This protocol has three versions, based on the amount of computation that each sensor must perform. The most computationally intensive predictive protocol is not scalable; we present it here as a benchmark for comparison. In the *complete-Voronoi protocol*, each sensor calculates every other sensor's motion and uses this to compute its Voronoi region after each event. This ensures the best performance because each sensor knows exactly what area it should consider for coverage. If any part of the sensor's Voronoi region is farther away than R_s, the sensor knows that no other sensor is closer to this point and that it should not move away from this point. (The sensor needs to check only the region's vertices, because the region is always polygonal.) As long as the sensor maintains its

TABLE 1
Properties of the communication-based algorithm (where *s* is the total number of sensors in the network, *n* is a sensor's number of neighbors, and *O* is the standard complexity measure).

Property	Value
Communication	$O(s)$ messages per event
Computation per sensor per event	$O(n \log n)$
Maximum range of neighbor knowledge	Communication radius; prone to double coverage
Connectivity	Guaranteed

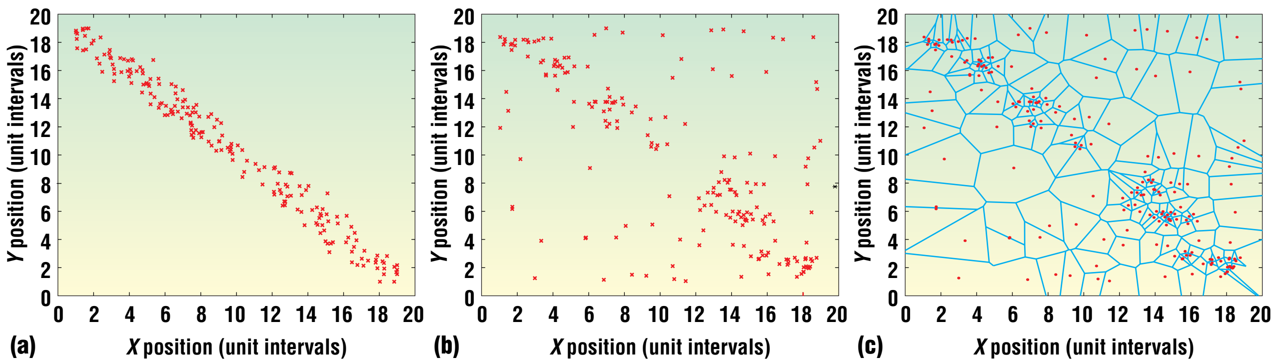


Figure 6. Representative results of predictive coverage maintenance: (a) event positions; (b) final sensor positions; and (c) a Voronoi diagram of sensor positions.

Voronoi region in this way, overall coverage continues.

Figure 6 shows a typical result of this technique. The sensors' Voronoi diagram shows no region larger than $R_s = 3$ units from each sensor's center.

Performing this prediction correctly involves a recursive problem: Once a sensor has stopped, it's no longer obeying the predictive rule. For a sensor to accurately predict the network state, it must also know which sensors have stopped. This can occur in two ways. If we desire no additional communication, each sensor can predict whether other sensors will stop on the basis of the same Voronoi region calculation. However, this is a very large computation, and we can easily avoid it with just a little communication. When one sensor stops to avoid coverage loss, it sends a broadcast message with the position at which it stopped. Other sensors can then assume adherence to the underlying motion algorithm unless they receive such a message. Because each sensor stops only once, only $O(s)$ broadcasts are required over the task's length, rather than s per event.

Table 2 lists the properties of the complete-Voronoi algorithm without and with communication.

Using the complete-Voronoi diagrams requires considerable computation, both to track all the sensors in the network and to compute the diagram itself. A scalable predictive protocol, the *local-Voronoi algorithm* trades off a little coverage accuracy for a large reduction in computation. After the initialization in which all sensors discover the location of all other sensors, each sensor computes its Voronoi region. As the task progresses, each sensor tracks only those sensors that were its neighbors in the original configuration. It then calculates its Voronoi region after each event on the basis of only this subset. It then examines its Voronoi region in the same way as in the complete-Voronoi protocol to determine whether to stop maintaining coverage. Table 3 lists this algorithm's properties.

As long as the neighbor relationships remain fairly constant, the local-Voronoi algorithm can produce results similar to those of the complete-Voronoi algorithm. In addition, the local-Voronoi algorithm makes sensors more conservative about coverage than the complete algorithm, because the calculated Voronoi region is based on a subset of the true neighbors and so can only be larger than the true region. When movement is small or generally in a single direction, the neighbor relationships remain fairly constant. If the motion is large or nearby sensors move in different directions, the neighbor relationships can change. In the latter case, we can modify the algorithm slightly by repeating the initialization step at regular intervals. This lets the sensors discover their new neighborhood, improving the algorithm's accuracy while still limiting communication.

Comparison

To compare the utility of these different protocols, we've conducted empirical

TABLE 2
Properties of the complete-Voronoi algorithm (where C_c is the amount of computation that the control algorithm requires).

Property	Without communication	With communication
Communication	None	$O(s^2)$
Computation	$O(s \log s) + O(sn)$ [coverage] + sC_c [prediction]	$O(s \log s) + sC_c$
Maximum range of neighbor knowledge	Arbitrary	Arbitrary
Connectivity	Not guaranteed	Not guaranteed

tests to determine the amount of communication and computation each algorithm requires under various circumstances. Because each protocol can work with either the history-free or the history-based update algorithms, we present the communication and computation required for the coverage-related portion. In the predictive algorithms, the computation amount depends on the update rule used. Table 4 presents the actual amount of computation used in the Matlab simulations for each algorithm.

The difference between the last two algorithms in Table 1 (namely, the use of occasional global-positioning updates) is only partially reflected in the communication and computation columns. Clearly, the periodic updates require additional communication, but the advantage to using this algorithm is that coverage detection is more accurate.

We can use the number of fixed sensors as a metric for comparing the algorithms. The rightmost columns in Table 1 list the number of sensors that the different algorithms require for coverage under three different event distributions. Because coverage was complete in all cases, the smaller the number here (meaning the fewer sensors required), the better. This shows that continual use of original neighbors is ineffective. Periodic updates of the neighborhood can give results that are almost as accurate as for the complete algorithm while using far less computation, and that use less communication than the communication-based algorithm.

One potential application of this work is in systems having many immobile sensors. Rather than all sensors being active at all times, a sparse set of sensors could be active and scanning for events. When events occur, different sensors could become active (and others inactive) to mimic the motion of sensors described in this article. This would allow the same concentration of active sensing resources while limiting the overall system's power consumption. The trade-off between using many immobile sensors versus fewer mobile sensors would then depend strictly on cost—mainly, the sensing elements' cost. Thus, costly sensors could be deployed on mobile platforms, and inexpensive sensors could be deployed on larger immobile systems.

We hope to develop other techniques for sensor positioning and extend our techniques to more complex tasks, such as constrained sensor motion and time-varying event distributions. From an algorithmic viewpoint, we could apply an approach similar to Kohonen feature maps, which use geometry to help classify underlying distributions. By defining the sensor closest to an event as the best fit to the data, we could update the neighboring sensors after each event. Rather than updating a virtual network's

TABLE 3
Properties of the local-Voronoi algorithm.

Property	Value
Communication	$O(s^2)$
Computation	$O(n \log n) + nC_c$
Maximum range of neighbor knowledge	Arbitrary
Connectivity	Not guaranteed

TABLE 4
Comparison of different coverage protocols based on Matlab implementations for common sets of 200 events of different event distributions in a network of 200 sensors. The rightmost columns give the number of sensors that each algorithm requires for each of the three different event distributions.

Algorithm	Communication (total no. of messages)	Computation (flops per event)	No. of fixed sensors Gaussian	Diagonal	Two lines
Communication-based (Figure 5)	s per event	65	94	62	36
Complete Voronoi without communication	s^2 initial	$40{,}000 + sC_c$	71	54	47
Complete Voronoi with communication	s^2 initial, $< s^2$ additional	$6{,}000 + sC_c$	71	54	47
Local Voronoi with no neighbor update	s^2 initial, $< s^2$ additional	$400 + nC_c$	125	156	123
Local Voronoi with updates every 20 events	s^2 per update	$400 + nC_c$	74	62	56

weights, the algorithm would simply change the sensor positions.

An example of a new application is one in which the environment has a complex shape or contains obstacles, or in which the sensors have particular motion constraints. In these cases, if knowledge of the constraints exists, the sensors might be able to plan paths to achieve their correct position, and the network could propagate this knowledge. The sensors could also switch roles if doing so enables more efficient behavior. Another important situation is one in which the event distribution changes over time. There are several different ways to let the sensors relax toward their initial distribution, and the best choice might depend strongly on the task and its temporal characteristics.

By developing algorithms for these situations, we hope to produce systems that can correctly react online to a series of events in a wide variety of circumstances.

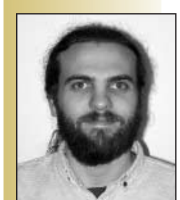
ACKNOWLEDGMENTS

We appreciate the support provided for this work through the Institute for Security Technology Studies; National Science Foundation awards EIA–9901589, IIS–9818299, IIS–9912193, EIA–0202789, and 0225446; Office of Naval Research award N00014–01–1–0675; and DARPA task grant F–30602–00–2–0585. We also thank the reviewers for their time and many insightful comments.

REFERENCE

1. A. Howard, M.J. Mataric, and G.S. Sukhatme, "Mobile Sensor Network Deployment Using Potential Fields: A Distributed, Scalable Solution to the Area Coverage Problem," *Proc. Int'l Conf. Distributed Autonomous Robotic Systems 5* (DARS 02), Springer-Verlag, 2002, pp. 299–308.

the AUTHORS



Zack Butler is a research fellow in the Institute for Security Technology Studies at Dartmouth College, where he is also a member of the Robotics Laboratory in the Department of Computer Science. His research interests include control algorithms for sensor networks and distributed robot systems, and design and control of self-reconfiguring robot systems. He received his PhD in robotics from Carnegie Mellon University. He's a member of the IEEE. Contact him at ISTS, Dartmouth College, 45 Lyme Rd., Suite 300, Hanover, NH 03755; zackb@cs.dartmouth.edu.

Daniela Rus is a professor in the Department of Computer Science at Dartmouth College, where she founded and directs the Dartmouth Robotics Laboratory. She also cofounded and codirects the Transportable Agents Laboratory and the Dartmouth Center for Mobile Computing. Her research interests include distributed robotics, self-reconfiguring robotics, mobile computing, and information organization. She received her PhD in computer science from Cornell University. She has received an NSF Career award, and she's an Alfred P. Sloan Foundation Fellow and a MacArthur Fellow. Contact her at 6211 Sudikoff Lab, Dartmouth College, Hanover, NH 03755; rus@cs.dartmouth.edu.