Motion-Constrained Mobile Sensor Networks

Zack Butler
Department of Computer Science
Rochester Institute of Technology
zjb@cs.rit.edu

Abstract— **The addition of mobility to sensor networks gives users the ability to efficiently monitor environments about which little is known ahead of time. For example, if there is a suspected path through the forest that should be monitored, but its exact location is not known, sensors can be scattered through the area and converge on the path as people or animals are detected on it. In this work, we present several distributed algorithms for sensors to produce this convergence. The first is a robust positioning technique for open rectangular spaces. We also build on this to produce an algorithm for arbitrary connected spaces. In addition, we consider sensors that are inherently constrained in their motion, such as tethered sensors fixed to a single line of motion. We present both analytical and empirical results for a variety of cases and discuss extensions that will provide greater effectiveness.**

I. Introduction

Sensor networks are increasingly important in monitoring dangerous or critical areas and providing large amounts of information to a central repository. Endowing networks of sensors with mobility can enable them to be more effective in monitoring environments. Sensors can move close to more active areas and redeploy as the situation changes. In particular, we are interested in the case where events tend to occur in the same portions of the environment, and we would like to have more sensors in those "hot spots" compared to the rest of the environment. If these hot spots cannot be known ahead of time, or if they change over time, we would like the sensors to discover them and gradually move toward them. Specifically, if the events in the environment come from a particular distribution, we would like the sensors to approach an approximation to that distribution.

In our previous work [1][2], we considered the basic case of a completely open, simply shaped environment, and developed distributed positioning and coverage algorithms. These algorithms used little or no communication among the sensors and produced globally correct sensor motion. Distributed coverage algorithms were overlaid on the basic positioning algorithms to guarantee that no portion of the environment would be left unsensed. However, these algorithms assumed that environment was simply shaped and that all sensors could move to any point in the environment. Here we extend these algorithms to more realistic systems; namely those in which the sensors have motion constraints, whether inherent or due to obstacles in their environment. This allows sensors to converge on event distributions in arbitrary connected spaces. We also present a new positioning algorithm which is more amenable to use in environments with obstacles, and can still be easily implemented.

While sensor networks have attracted a great deal of attention, mobile sensors have been investigated only recently. The addition of motion capability to the Mote sensors, creating Robomotes, was described in [3]. Early algorithmic work included even dispersal of sensors from a source point and redeployment for network rebuilding [4][5]. More recently, Bullo et al [6] developed Voronoi-based methods to arrange mobile sensors in various distributions in an analytic way that requires that the distributions be defined a priori. Another important consideration in sensor networks, whether mobile or not, is energy conservation. This has been investigated by Cerpa and Estrin [7] in traditional sensors and Rao and Kesidis [8] in mobile sensors. In the latter, sensors move so as to most efficiently relay data while producing sensor data of their own. Our work does not explicitly consider energy consumption, but is designed implicitly to produce limited motion and communication to achieve the desired formation. Our work is also similar to formation control (e.g. [9]) and related cooperative robotics tasks, but uses a common geometric algorithm rather than explicit planning and develops configurations specified entirely by sensed data.

II. Basic positioning algorithms

In previous work [1], we proposed two algorithms for sensors to independently determine their correct position based on sensed events. We summarize these here and present a new positioning algorithm. In each of these algorithms the goal is to have the sensors move so that their overall distribution will be similar to the distribution of events.

The first of these algorithms is purely reactive and heuristic, in that each sensor makes a short step toward each event when it occurs. The length of each step is determined by a simple function, and constraints can be placed on the form of this function to minimize clustering and produce reasonable sensor distributions. This method is desirable in its simplicity (memory and computational requirements) but can suffer from parameter sensitivity common to many heuristic algorithms. In this case, the sensitivity manifests in the sensors converging too quickly or too slowly on clusters of events. The second algorithm uses a coarse history of the events (in the form of a spatial cumulative distribution) to calculate a transform of the space, and from that determines the proper current position. This algorithm can be shown to be correct in the one-dimensional case, and is still quite simple. In two dimensions the transform is no longer one-to-one and a heuristic decision (in which the sensor chooses its best position from a small number of candidates) must be made. In practice, the sensor can simply choose the closest candidate. This works well for many types of distributions, but in some cases causes sensors to cluster overly tightly.

It should be noted that both of these algorithms are themselves deterministic, but their success comes in part from the assumption that the sensors are initially uniformly scattered over their environment. With this assumption, a transform of space based on the event distribution will allow the sensors to implicitly approximate this distribution without communicating their posi-

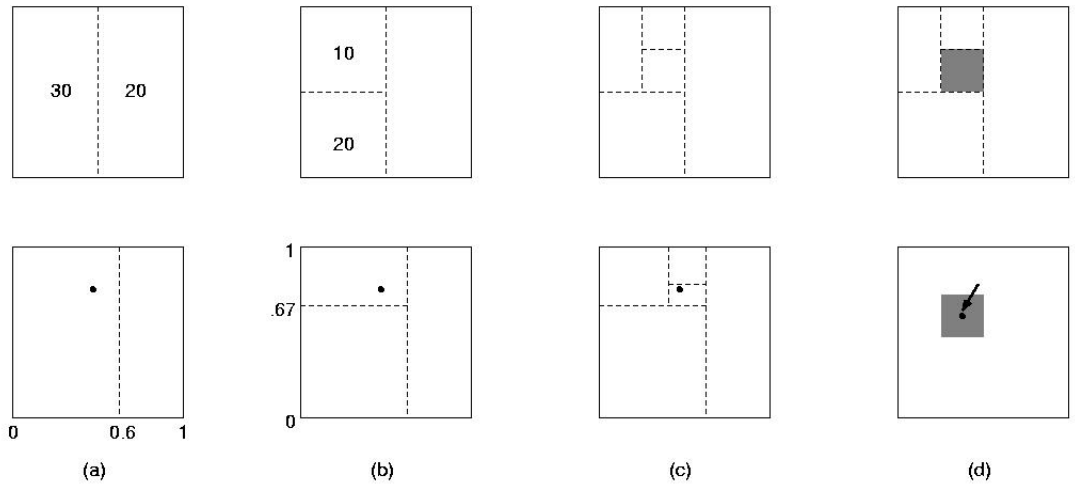

Fig. 1. Example run of the divide-and-conquer positioning algorithm, with the event histogram shown in the top row and the workspace in the bottom row. The sensor's initial position is shown as a black dot. (a) At the first step of the recursion, 30 and 20 events are counted in each half of the space, so the workspace is split accordingly. (b) The second step is to split the left half (containing the sensor's position). (c) After additional divisions, an area of the histogram corresponding to a single sensor is reached. (d) The sensor moves to the position in the environment corresponding to the relevant area in the histogram.

tions to each other. This assumption also allows the algorithms to be robust to noise in the positioning and motion of the sensors — if each sensor is out of place in a non-systematic way, the effective initial distribution of sensors will still be nearly uniform, and so there is graceful degradation of the final distribution as noise is introduced.

A. Divide-and-conquer positioning

We have also developed a new positioning algorithm which is one-to-one, simple and gives good results, although it is still dependent on an underlying coordinate representation. In this algorithm, which we refer to as a divide-and-conquer algorithm, we also use the event histogram to compute a transform between the space of initial positions and the desired positions.

As in the previous algorithms, each sensor uses a histogram of detected event locations and its initial position to independently determine where to move to. Here we assume that each sensor will be informed of all events, so that all sensors are using the same histogram in their computation. The algorithm computes a spatial transform recursively by successively dividing the space in half, first along one axis and then the other. This allows each sensor to compute only the portion of the transform that is currently relevant for its position. At each iteration, it divides the event histogram in half, but divides the workspace in two unequal rectangles that are sized proportionally to the number of events in each half of the event histogram. The recursion takes the portion of each space that corresponds to the sensor's initial position, and splits these along the other axis in the same fashion. The recursion terminates when the relevant portion of the workspace has an area proportional to $1/n$ of the total area (for n sensors), so that on average each sensor will choose a different area of the appropriate size. At this point, the sensor moves to a point within the remaining area of the event histogram. This is shown in pseudocode as Algorithm 1.

An example run of this algorithm for a workspace of unit size

Algorithm 1 Divide-and-conquer procedure. Desired position for a sensor is computed by calling COMPUTE-POS(overall-histogram-bounds, overall-workspace-bounds, x, normalized-histogram, my-initial-position).

COMPUTE-POS($hr, wr, axis, hist, ipos$)
  Variables:
    hr: region of $hist$ to be sensed
    wr: region of initial positions used to cover hr
    hr_i^*, wr_i^*: divided sub-regions of hr and wr
    c: number of events in each hr^*
  if axis = x **then**
    $hr_{1,2}^* \leftarrow$ SPLIT-RECT($hr, x, (hr.x_{min} + hr.x_{max})/2$)
  else
    $hr_{1,2}^* \leftarrow$ SPLIT-RECT($hr, y, (hr.y_{min} + hr.y_{max})/2$)
  $c_i = \sum_{hr_i} hist$
  total = $c_1 + c_2$
  if axis = x **then**
    midpoint = $wr.x_{min} + (wr.x_{max} - wr.x_{min}) * (c_1/total)$
    $wr_{1,2}^* \leftarrow$ split-rect($wr, x, midpoint$)
    if $ipos.x < midpoint$ **then**
      if $c_1 \leq 1$ **then**
        return position in hr_1^*
      else
        return COMPUTE-POS($hr_1^*, wr_1^*, y, hist, ipos$)
    else
      if $c_2 \leq 1$ **then**
        return position in hr_2^*
      else
        return COMPUTE-POS($hr_2^*, wr_2^*, y, hist, ipos$)
  Similarly for axis = y

SPLIT-RECT($rect, axis, value$)
  $r_1 = rect | axis \leq value$
  $r_2 = rect | axis > value$
  return r_1, r_2

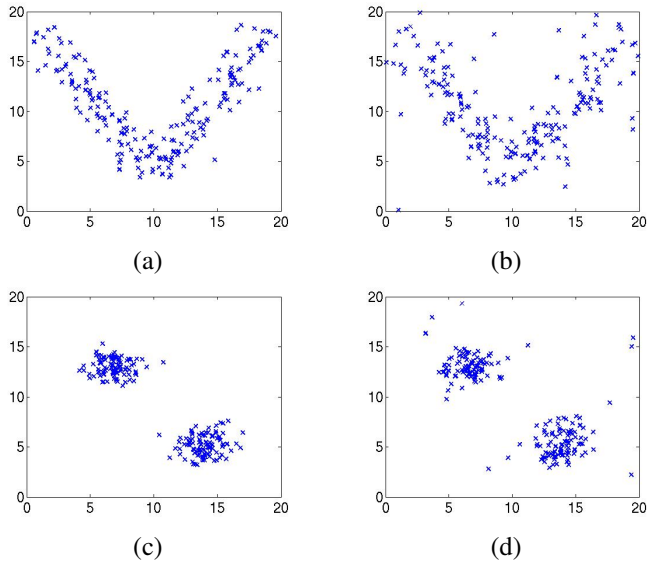

Fig. 2. Results of simulations using the divide-and-conquer positioning algorithm. (a) and (c) show event locations and (b) and (d) the corresponding resultant sensor positions.

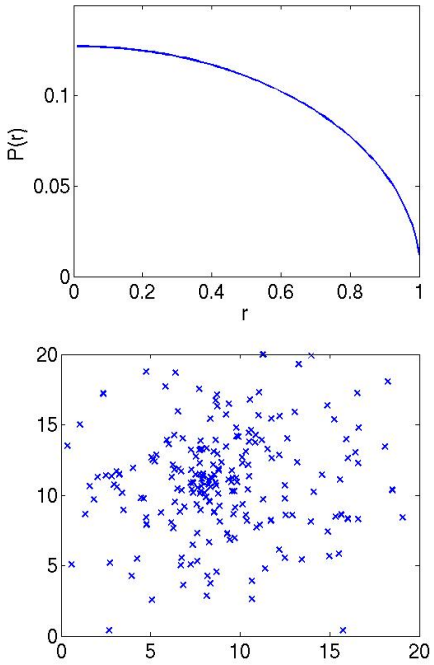

Fig. 3. (Top) Sensor distribution as a function of distance for sensors constrained to random lines and attracted to a single point. (Bottom) Simulated final locations of sensors restricted to lines and attracted to a single point.

is portrayed in Fig. 1. In the first iteration the algorithm divides the event histogram parallel to the y-axis into two equal sections. Here, the left histogram contains 60% of the total events, so the real space will be divided parallel to the y-axis into portions $[0, 0.6)$ and $[0.6, 1]$. Since the sensor's initial position has $x < 0.6$, the process is repeated with the left portion of the workspace and the left half of the event space. Since the splits of the event histogram may not fall along bin lines, the algorithm counts the fraction of the events in each bin corresponding to the fraction of that bin in the subset under consideration.

In practice, this produces accurate results with small amounts of sensor motion. Figure 2 shows two results of using this algorithm. Sensors are initially scattered through the environment with a uniform random distribution, and events occur at positions shown in Fig. 2a. The final positions of the sensors after all events have occurred is shown in Fig. 2b. The results of another run with a different event distribution is shown in Fig. 2c-d.

In these examples, the sensors closely approximate the event distribution, sometimes at the detriment of leaving portions of the environment empty. We have also developed algorithms under which the sensors cooperate to ensure that their entire workspace remains sensed[2]. These algorithms require only that each sensor can easily predict other sensors' motions. As long as the divide-and-conquer technique is implemented with each sensor using the same event history, since it requires only simple calculations based on a sensor's initial position, it is quite compatible with our existing coverage algorithms.

III. Sensor-based constraints

The positioning algorithms presented above assume that the sensors can move to any location within the environment. However, for many systems, this will not be the case. One potential situation would be tethered sensors with a limited range of motion. Another case that we have investigated more deeply is sensors that are constrained to work only along a line. These constraints are different from obstacle-based constraints detailed in Sec. IV in that each sensor has a unique restriction.

In order to correctly position motion-constrained sensors, we have several different options. One is simply to use one of the algorithms for unconstrained sensors to determine its desired location, and move the sensor to the closest possible point within its motion range. For well-clustered distributions, this may be a good choice. However, we have developed other algorithms specific to this case which have better empirical properties.

A. Analysis of linearly-constrained sensors

To determine the value of using the algorithm for unconstrained sensors for the constrained case, we can in fact analyze the overall behavior of the group, and in fact give a closed-form description of the group's distribution for simple cases. Namely, we assume a random distribution (for three different definitions of "random") of sensors and determine for a single given point of attraction, what is the resulting distribution of sensors? For the unconstrained case the sensors would eventually converge to a single point. For the constrained sensors, we can use a integral geometry argument to describe the resulting distribution.

To determine what distribution the constrained sensors would achieve in response to events at a single location, we note that the distribution will necessarily be radially symmetric and ask the question this way: For a given distance r, what fraction of sensors will move to a point within r of the event location? Equivalently, what fraction of sensors will have a line of motion that intersects a disk of radius r centered at the event location? Let us first assume that the sensor lines are given by a random point (ρ, θ) within a disk of radius R and a random orientation. The question is then, for a given point, what set of orientations

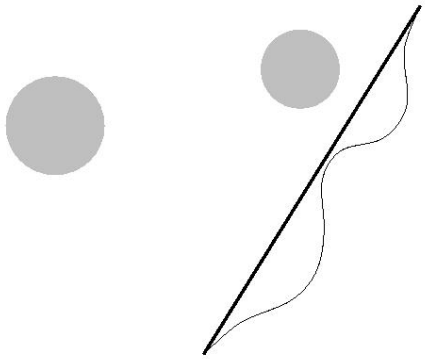

Fig. 4. Under the default algorithm, a sensor constrained to the thick straight line will be influenced by both clusters of events in its environment (as suggested by the curve below its line), but as it can only get close to one cluster it should probably ignore the other. For linear or other less-clustered distributions, the correct behavior becomes less obvious.

will allow the sensor's line of motion to intersect a disk of a radius r around the event. This is given by:

$$\Phi(\rho,\theta) = \begin{cases} 1 & 0 < \rho \leq r \\ \frac{\arcsin(r/\rho)}{\pi/2} & r < \rho \leq R \end{cases}$$

We can then integrate Φ over the entire disk to find the proportion of all sensors whose lines pass within the smaller disk. This final relation is given by:

$$F(r) = \frac{2r}{\pi R^2}\sqrt{R^2 - r^2} + \frac{2}{\pi}\arcsin\frac{r}{R}.$$

Similarly, we can choose sensor locations randomly by choosing two points within the disk, and consider the sensor's range of motion to be either finite or infinite and defined by these points. These lead to slightly different definitions of Φ, which can also be integrated to give similar but more complex analytic results.

$F(r)$ is a cumulative distribution, so the actual sensor distribution as a function of distance from the attractor, $P(r)$, will be its derivative. Plotting $P(r)$ for $R = 1$ gives a form as shown in Fig. 3, which is notably broad, i.e. even for events at a single point, constrained sensors will not produce a tight cluster. This is verified in simulation in the bottom plot of Fig. 3. With this in mind, we note that even for broad event distributions, it may not be possible to achieve desirable sensor distributions.

B. Event-scaling algorithms

For sensors constrained to a line, instead of trying to approximate an unconstrained sensor, we can also use a one-dimensional cumulative-distribution-based motion algorithm. In this case, all events that take place are projected onto the sensor's line of possible motion. The sensor then computes its desired position along this line based on its initial position along the line and the event distribution. (See [2] for more details on this algorithm.) However, this is not ideal, see Fig. 4, since some events may be closer to a given sensor's line of motion and therefore more relevant to it.

One obvious thing to do is to ignore events outside of a given distance from the line, but this leads to discontinuities in the sensor distribution when the events are not well clustered. Instead we can scale the events by distance, so that close ones count as full events while farther ones count as fractional events. In this way, if the sensor can move into a cluster of events, it will base its motion primarily on this cluster, but if it does not, it will move based on the cluster that is closest to its line of action. The scaling functions that were used were of the form $e = \min(k/d, m)$, where d is the distance from the event to the sensor's line of motion, and k and m are heuristic constants. The sensor then adds e events (where e is often less than 1) to its histogram in the appropriate bin before computing its new position based on the cumulative distribution of the events along its line of motion.

The scaling functions were compared empirically to the case without fading and the case where distant events are simply ignored. The success of a particular technique is defined here by how well the final sensor distribution approximates the sensed event distribution. To compare these distributions, we look at the number of events in each bin of the histogram (even where this is zero) and the number of sensors in the corresponding area of the environment. The histogram is scaled so that its sum is equal to the number of sensors, and the difference between the number of sensors and events in each bin is averaged. A low value of this mean deviation is desirable.

To generate empirical data, 50 simulation runs were performed for each scaling function with different initial sensor locations. As was expected, the default unscaled algorithms performed the worst, although ignoring distant events performed better under this metric than scaling. For events along a line through the workspace, the unscaled algorithm had a mean deviation of 1.416, compared to 0.908 for ignoring distant events and 1.111 and 1.144 for inverse distance functions with different constants. In all cases the standard deviations over the 50 runs was about 0.10, so these are significant differences. For comparison, unconstrained sensors were able to achieve a mean deviation of 0.580 for this event distribution. Similar results were obtained for events in two clusters similar to those seen in Fig. 2c: 3.073 for unscaled, 2.280 for the step, and 2.427 and 2.744 for inverse distance. Here unconstrained sensors had a mean deviation of 0.684, showing that for more clustered events, constrained sensors will perform relatively worse.

One drawback of this technique is that each sensor will maintain a different version of the event history. This will prevent the sensors from using a predictive coverage algorithm as presented in previous work. We have also developed more reactive coverage algorithms which could be used in this case[2], although these require a large amount of communication. In general, the issue of maintaining complete coverage with constrained sensors is an interesting topic in itself.

IV. Environmental constraints

Another important consideration in mobile sensor systems is that the environment or workspace will generally not be an simply-shaped open area. Therefore, we wish to augment our algorithms to handle more complex environments, starting with arbitrary convex regions and then allowing non-convex and non-simply-connected environments.

First, we note that although the positioning algorithms presented in Sec. II can be used in convex non-rectangular areas, the correct results will not be obtained. That is because bas-

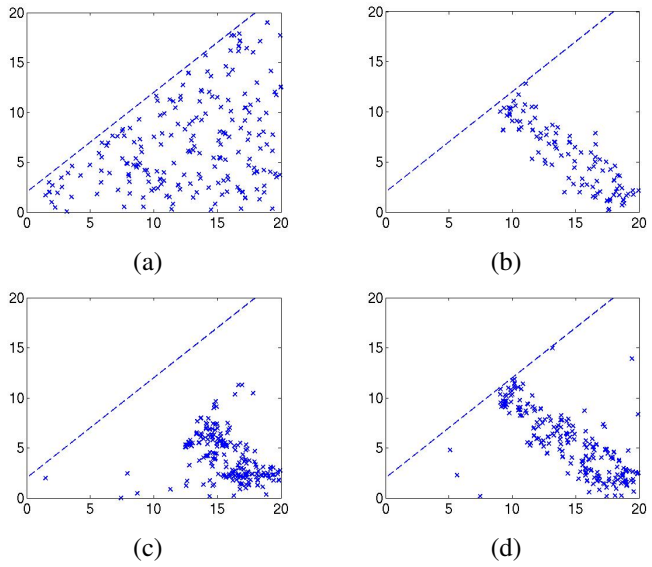

Fig. 5. Results of a simulation using the divide-and-conquer positioning algorithm in a non-rectangular environment. (a) Initial sensor positions. (b) Event positions. (c) Final sensor positions without accounting for environment shape. (d) Correct final positions.

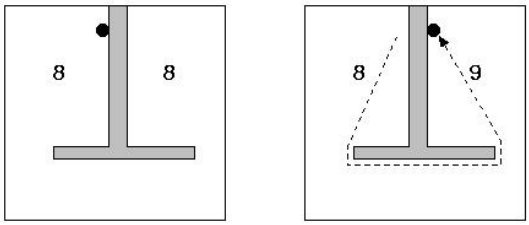

Fig. 6. In this example, (a) a sensor that started on the left side of the wall is there after 16 events — since half were on either side, it remains in the middle. In (b), after one more event on the right, it must move to the other side since the balance of the events are on that side and as the sensor in the middle it is responsible for any left-right imbalance. Another event on the left would send it back to the position in (a).

ing the sensor's desired position on its initial position within the overall rectangular histogram will bias the results. For example, consider the situation in Fig. 5. The histogram is the full size of the figure, but the sensors are all in the free space in the lower right. If this constraint on initial positions was not taken into account, the sensors would all think they started in the lower right half of their environment, and move to the lower right half of the event distribution as shown in Fig. 5c.

In the divide-and-conquer algorithm, the shape of the workspace is handled by changing how the workspace is divided in the recursion. Instead of dividing it along a line based solely on the proportion of events in each half, the shape of the workspace is considered. The division line is chosen such that the amount of free space in each half will be proportional to the number of events in the corresponding half of the histogram. Since the region is convex, the amount of free space to one side of a given line will be monotonic as the line moves, and so this division can always be made accurately and uniquely.

Non-convex and non-simply-connected environments give rise to further modifications to the basic algorithms. There are two natural ways to handle these cases in the context of the non-heuristic positioning algorithms. The choice between the extensions involves a tradeoff between inaccuracy in the distribution and requiring sensors to perform complex path planning.

In one method, we use the cumulative-distribution-based positioning algorithm and simply keep at zero the histogram values for any bin that contains part of an obstacle. This works immediately, since it enforces that no sensor should attempt to achieve a position within the obstacle, but the underlying function is still well-defined over the entire workspace. It is also immediately reactive in that when a new obstacle is detected, its bin of the event histogram can be zeroed and the sensors will immediately position themselves correctly. Depending on the granularity of the histogram, this may be overly conservative. One potential issue with this method is that when a sensor computes its new position, it may have to go to the other side of an obstacle. In an extreme case with two rooms and events alternating between the two rooms, such as in Fig. 6, a sensor in the middle of the environment could have to switch rooms after each event.

The alternative method is to decompose the environment into a set of convex regions and run a basic positioning method internal to each region. This requires the computation of such a decomposition (and recomputation if obstacles are later discovered), but has the advantage that any correctness properties of the underlying positioning algorithm will carry over locally. That is, within each region the relative sensor density will match the event density. However, globally this will not be the case — if one region has many sensors but no events, the sensors will not move, even if another region sees many events and few sensors. Therefore, we need to enable reassignment of sensors between regions.

To generate a reassignment, we first determine how many sensors should be in each region of the environment, and then build a cost matrix based on the distance from each sensor to each region. The assignment problem (in this case, of sensors to regions) is solved using the traditional Hungarian algorithm. In this particular application, the rows of the cost matrix correspond to sensors, and the columns to destinations, with the value in the matrix cell equal to the distance between the sensor and the destination. The result is a reassignment that puts the correct number of sensors in each region with the minimum overall travel (based on the particular distance metric used).

To create the cost matrix, we need to have a number of destinations equal to the number of sensors. For each region r, we scale the number of events in that region E_r to determine the number of sensors S_r that should go into that region. There will then be S_r identical destinations (and therefore identical columns in the cost matrix) corresponding to region r. Also note that each sensor currently in a particular region should be treated equally, to preserve the assumption of a uniform initial distribution — if we moved the nearest sensors from one region into an adjacent region, the remaining sensors would be imbalanced within their region. Therefore, in the cost matrix, each sensor's distance to each destination is computed solely as the number of regions the sensor would pass through to reach the destination region. This produces a random correct assignment with minimal region traversal. When a sensor is reassigned, it

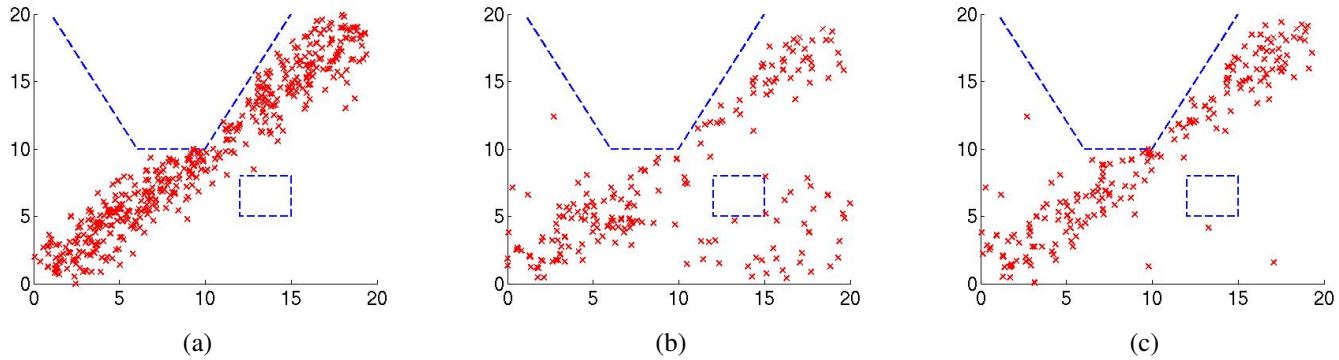

Fig. 7. Effects of reassigning sensors between regions: (a) Event positions (b) final sensor positions without reassignment, (c) final sensor positions with reassignment every 28 events.

picks a new "initial position" randomly from the free space of its new region. It does not go to this position, but rather uses it to determine its new desired location. Note that due to the nature of the divide-and-conquer positioning algorithm, it is not necessary to inform the other sensors of a reassignment or new initial position unless an additional coverage algorithm is in use.

Simple path planning is still required, however only between adjacent convex regions, which can easily be achieved by moving to a point on the boundary of the two regions and moving to the new point in the new region. Both of these motions are guaranteed to be navigable with straight lines since both regions are convex. With a decomposition into convex cells even arbitrary path planning can be easily achieved with graph search, but using local positioning with region reassignment will minimize the number of long excursions in complex environments.

Figure 7 shows one example of using reassignment in a moderately complex environment. The positions of 200 events are shown in Fig. 7a, with obstacles outlined by the dashed lines. Without reassignment, the sensors move somewhat reasonably, but a large number in the lower right are stuck in their (rather boring) region. In Fig. 7, we show the results of reassignment every 28 events. In this example, a total of 84 sensors out of 200 changed regions a total of 106 times. In another case, we generated a tight cluster of events in one corner of the same environment. In this case, 55 of the 200 sensors started in the region containing all the events, and by the end, 192 of the 200 were there. This included a total of 190 sensor motions (by 138 sensors), so this algorithm produced very efficient overall reassignment. This efficiency is largely due to the stationary distribution assumption, but does show that the algorithm is stable.

V. Discussion

The algorithms presented in this paper provide a step toward useful real-world networks of mobile sensors. Systems can now generate reasonable sensor distributions over a wide variety of environments, although the method of decomposing the environment was not specified here.

These techniques still implicitly assume a stationary distribution of events (one that does not change over time), in that any event added to the histogram stays there forever. We have investigated different ways to handle time-varying distributions, such as rescaling the histogram over time to give more recent events greater weight, or adding a small virtual force between the sensor and its initial position. These can each be successful, but are heuristic in nature, and the quantitative nature depends on the particular situation. We are now working on meta-heuristics; that is, how to watch the event distribution evolve and feed this back onto the heuristic values in the time-dependent algorithm. This strategy would make better use of the sensors' mobility in that it would more accurately and automatically track situations that change over time.

Acknowledgments

Much of this work was performed at the Institute for Security Technology Studies (ISTS) at Dartmouth College. Thanks to Scot Drysdale and Amit Chakrabarti for helpful discussions on the statistics of linear-motion sensors. Daniela Rus was instrumental in suggesting and encouraging this work. This work was supported under Award No. 2000-DT-CX-K001 from the Office for Domestic Preparedness, U.S. Department of Homeland Security. Points of view in this document are those of the authors and do not necessarily represent the official position of the U.S. Department of Homeland Security.

References

[1] Z. Butler and D. Rus, "Event-based motion control for mobile-sensor networks," *Pervasive Computing*, vol. 2, no. 4, pp. 34–43, 2003.

[2] Z. Butler and D. Rus, "Controlling mobile sensors for monitoring events with coverage constraints," in *Proc. of IEEE ICRA*, April 2004.

[3] G.T. Sibley, M.H. Rahimi, and G.S. Sukhatme, "Robomote: A tiny mobile robot platform for large-scale sensor networks," in *Proc. of IEEE ICRA*, 2002, pp. 1143–8.

[4] M.A. Batalin and G.S. Sukhatme, "Spreading out: A local approach to multi-robot coverage," in *Distributed Autonomous Robotic Systems 5*, 2002, pp. 373–382.

[5] 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," in *Distributed Autonomous Robotic Systems 5*, 2002, pp. 299–308.

[6] J. Cortes, S. Martinez, T. Karatas, and F. Bullo, "Coverage control for mobile sensing networks," *IEEE Trans. on Robotics & Automation*, vol. 20, no. 2, pp. 243–55, 2004.

[7] Alberto Cerpa and Deborah Estrin, "Ascent: Adaptive self-configuring sensor networks topologies," in *INFOCOM*, New York, NY, June 2002.

[8] R. Rao and G. Kesidis, "Purposeful mobility for relaying and surveillance in mobile ad hoc sensor networks," *IEEE Trans. on Mobile Computing*, vol. 3, no. 3, pp. 225–32, 2004.

[9] Tucker Balch and Maria Hybinette, "Social potentials for scalable multi-robot formations," in *Proc. of Int'l Conf. on Robotics and Automation*, San Francisco, April 2000, pp. 73–80.