Light-weight Contour Tracking in Wireless Sensor Networks

Xianjin Zhu* Rik Sarkar* Jie Gao* Joseph S. B. Mitchell†

* Department of Computer Science, Stony Brook University. {xjzhu, rik, jgao}@cs.sunysb.edu
† Department of Applied Mathematics and Statistics, Stony Brook University. jsbm@ams.sunysb.edu

***Abstract*—We study the problem of contour tracking with binary sensors, an important problem for monitoring spatial signals and tracking group targets. In particular, we track the boundaries of the blobs of interest and capture the topological changes as the blobs merge or split. Only the nodes on the boundaries of these deformable blobs stay active and the repair cost is proportional to the size of the contour changes. Our algorithm is completely distributed, requires only local information, and yet captures the global topological properties. The algorithm performs a fundamental monitoring function and is a foundation for further information processing of spatial sensor data.**

I. Introduction

One of the most promising applications of wireless sensor networks is distributed sensing and information gathering. Thanks to the relatively low cost of sensor nodes and the ease of deployment with wireless communication, distributed sensing, in comparison with traditional centralized sensing (such as weather stations and satellite remote sensing), is able to provide a level of fine sensing resolution that was not achieved before and is expected to become a fundamental driving force to expedite scientific research and enable novel applications [1], [2]. In the past few years there has been a tremendous amount of work on networking sensors and signal processing in support of information gathering and environment monitoring applications.

Most physical measurements exhibit strong spatial and temporal correlations, since physical phenomena are predominantly governed by the law of diffusion. In this paper, we study the problem of tracking contours represented by binary sensors, and we focus on light-weight maintenance of contours that evolve over time. This abstracted problem is motivated by a variety of tracking and monitoring applications.

Contour tracking scenarios. Consider an application scenario in which the sensors are used to detect and track chemical pollution. Each sensor measures the chemical intensity in its vicinity. As chemical contamination often comes from some pollution source, and the propagation of contaminants is typically by water current, wind, or diffusion, the pollution map exhibits strong spatial correlation and is often modeled and represented by a smooth signal field. The contaminated regions, having sensor readings above a danger threshold, naturally form a number of (possibly nested) blobs. Over time, the blobs may morph, merge, or split, indicating the pollution movement and/or the effectiveness of pollution treatment.

In another example, a group of targets moving in a field may alert the monitoring acoustic sensors nearby. Target movements in nature, such as human, vehicle, animal movements, have a tendency to be clustered. A group target can be monitored by tracking the contour of acoustic readings above a certain threshold. Contour changes reveal important information, e.g., the formation of a team or gathering, the dispersion of vehicles, or certain animal activities.

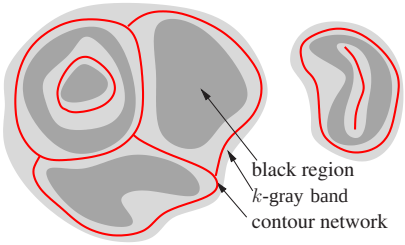


Fig. 1. An example of the BLACK regions, k-gray band, and the contour network.

Contour tracking can also be applied to monitoring the health of the network itself. A well-behaving network should avoid traffic congestion and unbalanced energy depletion. Each node can locally determine its "health level" based on its traffic load and energy reserves. A contour tracking protocol identifies the congested regions and low-energy regions, providing the user a global view of network health.

Note that, in all of the above application scenarios, simply detecting nodes around the boundary of the evolving blobs is not sufficient. There can be a large number of nodes within a thick band identifying themselves on the boundary based on local readings, but none of them has a clear idea of the global picture of the entire signal filed — the number of disconnected pieces, the merging/splitting of them, the nesting relationships, etc. Those topological features of these contours are often of special interest to users. When monitoring contaminants, a user may query for a low-risk path through the geographical domain of the sensor field; if completely surrounded by hazardous chemicals, then special care (e.g., a rescue helicopter) may be needed. In tracking group targets, we may want to ensure that the targets do not surround a certain landmark. In monitoring network health, we want to make sure the network remains connected.

The challenges of contour tracking. This paper presents a distributed algorithm that maintains, on the fly, the contours and their topological changes, while minimizing the use of network resources. We focus on the maintenance of contours of a single level-set (as in the scenarios above); the algorithm can be applied to the multi-level contours case and, potentially, to learning topological features of the sensor field.

We first survey previous work on tracking and contour detection and then explain the challenges of light-weight contour

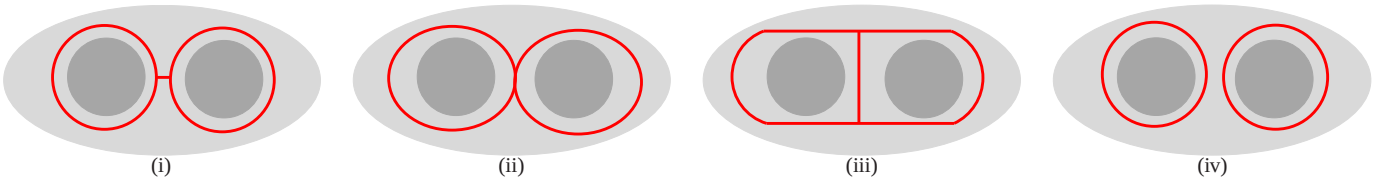


Fig. 2. A field with two blobs. Figure(i)-(iii) show three valid contour networks of the gray band – all of them are deformation retract (intuitively, they all capture the fact that the gray band is connected and has two holes) but have different local features. Figure (iv) shows an invalid contour network. It is hard for an individual sensor to figure out the global topology.

tracking.

Most prior work on tracking by distributed sensors focused on tracking of individual targets (e.g, vehicles, humans, animals) moving in the field (see [3]–[8] and references therein). Tracking of a continuously deforming blob or of groups of targets is not as well studied. One could apply existing target tracking algorithms to each individual in the group, but this is not only *inefficient*, since only the sensors near the boundary of a possibly large blob need be involved, but also *insufficient*, since important topological changes (e.g., splitting and merging of blobs) are not easily available. Liu *et al.* [9] studied the problem of tracking a half-plane shadow by using geometric duality, exploiting the continuity of the contour and identifying the "frontier" sensors that may be included in the shadow in the future. Their approach, however, requires node locations, centralized pre-processing, and non-local communications.

In the static scenario, when the targets do not move or when considering a snapshot of chemical spreading, the problem reduces to detecting the boundary or the "holes" in the network, where sensors have abnormal readings. There has been a lot of work on boundary detection [10]–[17]. In the dynamic setting, one can periodically run a hole detection algorithm to discover contours. However, a major issue is choosing the update frequency — frequent updates waste network resources and infrequent updates miss critical changes. The update frequency is often dictated by the highest frequency of changes in the contours. Further, periodic update schemes requires global coordination and good network synchronization.

The problem we want to tackle here is to construct and maintain a *contour network* that abstracts the global topology of the contour components. This is very tricky since the topology of the contours represents a global feature of the signal field, and thus, an individual sensor cannot easily tell the computed contour network is valid or not (see an example in Figure 2). Our goal is to devise an algorithm in which each node maintains some local states, yet collectively they accomplish the global task.

Our contribution. We propose a light-weight and distributed algorithm to track the contours as they evolve over time. We construct and maintain a *contour network*, which tightly surrounds the contours and captures precisely the important topological features, e.g., how many connected components and how the contour components are connected and nested. See Figure 1 for an example.

As the contours evolve, the basic idea is to freeze the valid segments in the old contour network, and only repair the contour network where it is broken. We propose a local algorithm such that, within only a local neighborhood of the broken contour, a node without the global knowledge can still repair the contour network and maintain the topological properties. Our algorithm has the following characteristics.

- the topology of the contours is captured precisely;
- the communication cost is "output-sensitive", being proportional to the magnitude of the changes to the contours, with the algorithm adapting automatically to the frequency and intensity of changes in the input data; and,
- the algorithm requires only local communication and does not require node location information.

The light-weight contour tracking algorithm serves as a fundamental network monitoring module, providing the basic input for further processing and representation of the signal field, e.g., for contour aggregation and simplification [18], [19]. It also allows efficient use of system resources, since the nodes not in the vicinity of the contour can stay on low duty-cycle and thus reduce energy consumption.

We present in Section II a distributed and practical implementation of the algorithm. Augmented with an additional process, we provide theoretical guarantee on the contour network property. We include the proofs in Section III.

II. Contour Tracking Algorithm

A. Problem Setup

We consider a set of sensor nodes densely deployed in an environment of interest. We abstract the problem by assuming a continuous signal field σ covering the entire domain. Each sensor i has a reading, which is a discrete sample of the signal field at the location of i, measuring the value of a physical phenomenon being monitored. In this paper, the sensors are static, but the signal field evolves over time. For description simplicity, we focus on single-level contour tracking; i.e., there is a certain predicate that specifies the range of sensor readings of interest to us, and we define for each sensor a 0/1 variable called the "contour value" of the sensor.

Definition 2.1. ***Contour value****: The contour value $v(i)$ of node i is set to 1 if the reading of node i is within the range of tracked contour levels, and 0 otherwise. The set of contour values within the 1-hop neighborhood of node i (including i itself) is denoted by $V(i)$.*

A practical concern of the contour tracking algorithm is to deal with the issue of robustness of using the discrete values to approximate the continuous signal field. In particular, our algorithm only keeps track of contours of "sufficient significance", which is formulated in terms of colors:

Definition 2.2. ***Color****: The color $c(i)$ of node i is defined as:*

- BLACK*: $0 \notin V(i)$, i and all of its neighbors have value 1.*
- WHITE*: $1 \notin V(i)$, i and all of its neighbors have value 0.*
- GRAY*: $0 \in V(i)$ and $1 \in V(i)$, a node that is neither* BLACK *nor* WHITE*.*

Thus, to enhance the robustness of the system we keep track of the connected components of BLACK nodes, called *black regions*. This introduces two benefits. First, a collection of sensors with "salt-and-pepper" type of contour values do not have significant contours to be tracked by our algorithm. Thus, we are more robust to noises in sensor measurement: A single value 1 in a group of sensors with value 0 does not trigger contour creation; rather, only when there is sufficient evidence witnessed by a node and all of its neighbors having value 1 does it indicate the node is BLACK and there is a contour worthy of tracking. Second, a BLACK node cannot be adjacent to a WHITE node, by definition. There is a GRAY band that separates the BLACK regions from WHITE regions. We are interested in learning and maintaining the shape and topology of the BLACK regions; symmetrically, we can track the WHITE region by reversing the contour values. Our algorithm outputs a *contour network* inside the GRAY band and tightly surrounding the BLACK regions as they evolve over time.

Definition 2.3. ***Contour network****: A network of* GRAY *nodes within k hops from* BLACK *regions. Nodes of the contour network are called* RED *nodes.*

All of the contour tracking operations are performed on the GRAY nodes within k hops from BLACK regions, denoted as *k-gray band*. k here is a parameter that characterizes the tightness of the contour network in approximating the boundaries of the BLACK regions. See an example in Figure 1. We set k to 2 in our simulations.

In this section we will describe a practical algorithm for contour tracking in a distributed network. An augmented version of the algorithm, described in the next section, can be proved to maintain a contour network with the same topology as the k-gray band.

B. State Transitions

Our contour tracking algorithm is abstractly thought of as an automaton running at every sensor node. The state transition of the automata is based on the states of k-hop neighbor nodes, and does not require location information. The color of node i is the "state" of the automaton running at node i (see Figure 3). Information stored at each node is minimal, including its node ID, contour value, color, and RED neighbors on the contour network.

Node i notifies all of its 1-hop neighbors about the change of its contour value $v(i)$; and notifies its k-hop neighbors about the change of its color/state. All transitions happen when $v(i)$ changes or node i receives notifications from neighbors. At states S3 (GRAY) or S4 (RED), node i tracks its neighborhood to decide whether to stay at current state or change to BLACK/WHITE. If node i finds at least one neighbor with different contour value from itself, i remains in current state; otherwise it transits to S1 or S2 depending on $v(i)$. To do this, a node maintains two counters recording the number of neighbors with contour value 1 and 0 respectively, and decrease/increase the corresponding value as informed by its neighbors. When a RED node i turns to BLACK/WHITE, i must leave the contour network, which may result in a broken contour. Therefore, the transition from RED to BLACK/WHITE triggers contour repair. On the other hand, if new GRAY nodes find that they are close to BLACK nodes but cannot see RED nodes nearby (within its k-hop neighborhood), which is possibly a sign of the appearance of a new BLACK region, those GRAY nodes would start to collaboratively construct a new contour. The operations of contour creation and contour repair are atomic operations that are not interrupted by other transitions. Nodes are locked when they enter into either of these two phases until the repair/creation is finished. In the following, we discuss the details of contour repair.

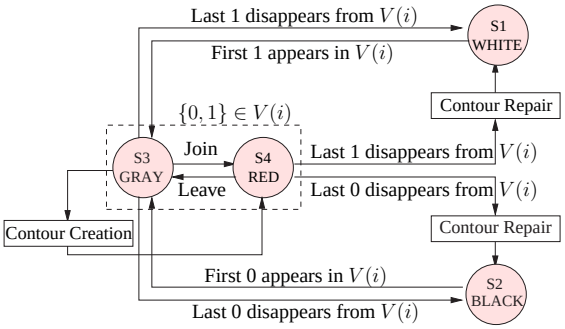


Fig. 3. State transitions of the automata running at each node.

C. Contour Repair

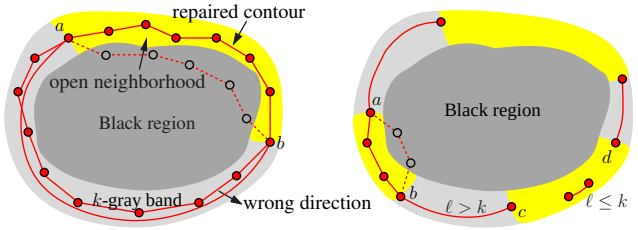
There are different cases that require contour repair — a single contour moves, expands, shrinks, splits, or multiple contours merge to one. Because every node can only see the changes within its local neighborhood, a sensor node has absolutely no idea of the global changes of the contour topology. Therefore, the major challenge in contour repair is to reconnect the broken contours in a distributed fashion, such that the resulting contour network is still topologically valid. In the following, we first describe the scenario of repairing a single contour cycle, then move to the cases of simultaneous repair of multiple contours, contour merging and splitting.

1) Repair of a single contour cycle: When a set of RED nodes leave the contour cycle (in the case of Figure 4 (i) these nodes change to BLACK), this leaves part of the contour cycle invalid and a few nodes with a missing RED neighbor. We call these nodes as *open* RED *nodes*, such as a and b.

Definition 2.4. ***Open* RED *Nodes & Closed* RED *Nodes****: When a* RED *node loses (one or more) of its current* RED *neighbors, it becomes open. Otherwise, it stays as closed* RED *node. A* RED *node may also become open if triggered by others. An open* RED *node actively repairs the broken contour.*

When a RED node first becomes open, it is responsible for repairing the broken cycle. It initiates a repair message within the k-gray band, looking for other (open or closed) RED nodes to connect to. However, without location information a sensor node lacks the sense of directions. If we allow the repair

message to propagate freely in the k-gray band, the resulting cycle may not be valid (as shown in Figure 4(i)). Therefore, it is important to block traffic traveling in the wrong direction, implied by the closed RED neighbors of the open RED node.



(i) (ii)

Fig. 4. Repair of a single contour. Open neighborhoods are highlighted. (i) a and b are two open RED nodes. The repair message only travels within the open neighborhood. (ii) A single contour cycle is broken into multiple chains.

Definition 2.5. ***Blocked neighborhood and open neighborhood****: a* RED *node is defined to be a* block node *if it is closed and is at least k-hops away from an open* RED *node in a connected component of a contour cycle. The union of the k-hop neighborhoods of block nodes is the* blocked neighborhood B*. The rest of the k-gray nodes is called the* open neighborhood $O = \mathcal{G} \setminus B$*. The open neighborhood has a number of disconnected components, denoted as O_i.*

Clearly each connected open neighborhood O_i has at least one open RED node. The repair message only travels within the open neighborhood, as shown in Figure 4 (i). Since the repair operation is confined inside each O_i there is no interference between the repair efforts in different connected components.

Notice that for the two open nodes a, b, without the knowledge of each other, each would attempt to repair the contour. We suppress the repair messages by the ID of the initiator. In particular, the repair message will not be forwarded when it either (i) enters the blocked neighborhood, or (ii) arrives at some nodes who have received repair messages initiated by other open RED nodes with ID *lower* than its initiator. As the repair message is forwarded, an aggregation tree rooted at a is also cached on the nodes who forward the message. This aggregation tree helps the open RED nodes gather information about RED nodes encountered on the way. In particular, when a node stops forwarding, it returns with the ID of the RED nodes it learned so far. At an internal node of the aggregation tree, a node propagates up the aggregated information when all of its children have reported their information. If an open RED node learned through the information gathered that there is another open RED node with smaller ID, then it retires. The node with smaller ID, called the *repair node* for this open neighborhood, is responsible for the repair and selects a path connecting to the other open RED node. The GRAY nodes on this path are invited to join the cycle and turn to RED.

In general, a RED cycle may be broken into multiple chains (Figure 4(ii)). The repair is done in a similar way. All of the nodes with at least k hops away from an open RED node block their k-hop neighborhood. If a left-over contour chain is shorter than k, it is hard to distinguish the correct repair direction from the wrong direction at open RED nodes. Those short chains become un-repairable.

For un-repairable chains the open RED nodes find they are on chains with length $\ell < k$. They remove themselves from the contour network and turn back to GRAY. Eventually the entire short chain disappears. Remaining repairable chains still participate in cycle repair (Figure 4 (ii)). When a RED cycle is broken into all short chains, it is good time to discard all un-repairable chains and reconstruct a new cycle from scratch using cycle creation algorithm, to be explained later.

2) Simultaneous repair, merging and splitting of BLACK *regions:* When multiple BLACK regions are close, their repair processes may interfere with each other. An open RED node may see multiple open RED nodes. This typically happens when the topology of the BLACK regions changes, i.e., two BLACK regions merge together or one BLACK region splits into several, the contour network will need to capture the new topology.

When a RED node a first becomes open, again it initiates a repair message which is propagated in the k-gray band as before (with the k-th hop closed RED neighbor of a blocking the message propagation). We focus on the open neighborhood O_i containing a. If there are multiple open RED nodes in O_i, again the repair messages from all but the one with lowest ID are suppressed. The repair message travels to every node in an open neighborhood O_i and is only stopped if it hits the BLACK regions, or the boundaries of the k-gray band, or blocked by the k-hop node of some block nodes. Thus a learns about these block nodes, which are grouped into *bounding segments*.

Definition 2.6. ***Bounding segments.*** *We take the* RED *nodes within k-hop of an open neighborhood O_i and denote them the* bounding RED nodes *of O_i. Each connected component of the bounding nodes is called a* bounding segment.

A bounding segment may or may not have an open RED node. See Figure 5 for two examples. Now the repair node a simply connects through shortest paths to each and every bounding segment.

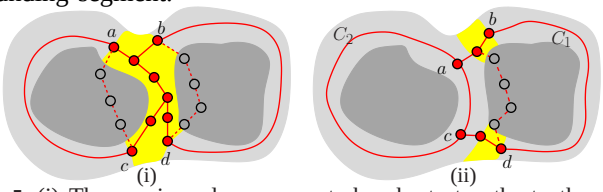


Fig. 5. (i) The repair node a connects by shortest paths to the other bounding segments in its open neighborhood. (ii) The open RED nodes b, d connect to their respective bounding segment (in this case, a segment with only closed RED nodes).

In a special case, an open RED node may not see any bounding RED node in a different segment than itself, the contour repair operation gets stuck. Figure 6 shows two such scenarios. The open neighborhood of node b in Figure 6 (i) has only one bounding segment containing itself. In this case the repair operation at b will terminate and node b stays as it is. The next node adjacent to b, in this case, node c, will now become an open node and attempts to repair the contour. Figure 6 (ii) shows a case when both a and b can not discover any RED nodes other than those connected to them. This will eventually

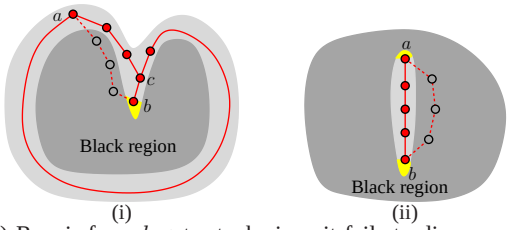


Fig. 6. (i) Repair from b gets stuck since it fails to discover any other bounding segment to connect to. Node b becomes closed and triggers the next node to be open. (ii) Both open RED nodes a, b fail to repair and become closed. The adjacent node is triggered to be open.

leave a RED segment as part of the contour network.

It is possible that some nodes between nearby red chains change states and become k-gray. In such a case the red chains may need to be connected or merged together to reflect desired homotopy. As such, a simple check for presence of red nodes within the k-neighborhood does not always suffice. So, a newly turned k-gray node does the following. It looks at its connected k-gray neighborhood (by initiating a k-hop flood and aggregation), and verifies that the red nodes in this neighborhood form a tree. If they do not, then all these red nodes are eliminated and become open. Open red nodes start contour repair as described above.

D. Contour creation and disappearance

Initially, as a BLACK region appears and grows, the creation of a new contour is triggered at some BLACK nodes that have a GRAY neighbor but cannot see RED nodes in its k-hop neighborhood, because this indicates the appearance of a new BLACK region not tightly surrounded. The BLACK node turns its GRAY neighbor to an open RED node.

Now a GRAY node i enters *contour creation* phase. It is possible that multiple GRAY nodes try to create a new contour for the same BLACK region. To avoid every GRAY node repeating the same thing, the GRAY nodes who want to start a contour will participate in a local leader selection procedure. The leader selection algorithm selects a node as a leader if no other node within its k-neighborhood becomes a leader. This can be done with any clustering algorithm or by local message suppression. After that, only leaders participate in the creation. The distance between any two leaders is at least k hop apart.

The leaders become open RED nodes. But we consider the k-hop neighborhood of a leader to be blocked to a different leader node. Now the leaders use the contour repair algorithm to find other leaders to connect to. The repair messages from leaders will meet either other bounding RED nodes or nodes with repair messages from other leaders/repair nodes. In both cases, the leader with smaller ID will be selected to build a path to connect to the desired party. GRAY nodes on the path will turn to RED too and together with the leaders form a red chain with length at least k. With the same contour repair protocol these chains will eventually be connected to a contour network.

The contour network disappears as the corresponding black regions disappear, because open RED nodes that are supposed to repair the cycle will detect that they are not within k-hop of any BLACK node. Those RED nodes would remove themselves from the contour automatically, and the contour disappears eventually.

E. Summary of the repair algorithm

The general principal of contour repair is that the valid segments of old contour network is still usable and repair only happens where the contour is broken. The repair node connects through a *spanning tree* to all other bounding segments of its open neighborhood.

We also emphasize a few issues in the implementation for algorithm robustness: (1) The contour network in an open neighborhood O_i is repaired by the open RED node with the minimum ID in the bounding segments. Thus even in a distributed setting there is always consistency agreed upon who makes the decision. (2) When a node is involved in a repair procedure, it is temporary locked and does not participate in other repair procedures. When the repair operation terminates, the repair messages and the nodes who forwarded them will be refreshed. (3) The repair effort is confined within the open neighborhood, whose size is proportional to the contour changes.

In most common scenarios, when there are no small black regions that newly show up in the k-gray band, it can be proved that the proposed repair algorithm generates a contour network that is a deformation retract of the k-gray band. Intuitively this means that the contour network is a proper "thinning" of the k-gray band, capturing all of the topological information of the band, e.g., how many holes there are and how they are connected/nested. We handle the small holes in Section III and give a rigorous proof of this homotopy equivalence property.

III. MAINTENANCE OF TOPOLOGICAL FEATURES

The previous section focuses on the practicality and implementation details for contour tracking. The main contribution in this section is to provide a theoretical understanding of contour tracking. In the proofs we consider a continuous setting in which there are BLACK regions and WHITE regions in $\mathbb{R}^2$, separated by a GRAY band. The k-gray band $\mathcal{G}$ contains the GRAY points within distance k from BLACK regions. The contour network is denoted by G; an algorithm invariant is that $G \subset \mathcal{G}$. Our main theoretical result is an algorithm (an elaboration of that of the previous section, handling small holes inside $\mathcal{G}$) that, upon stabilization (when nodes no longer change state) computes a contour network with precisely the same topology as the k-gray band.

Rigorously, we show that the contour network G is a deformation retract of the k-gray band $\mathcal{G}$. A continuous map $\pi : \mathcal{G} \times [0,1] \to \mathcal{G}$ is defined as a *deformation retraction* if, for every x in $\mathcal{G}$, a in G, and $t \in [0,1]$, $\pi(x,0) = x$, $\pi(x,1) \in G$, $\pi(a,1) = a$. A deformation retraction is thus a homotopy between the identity map on $\mathcal{G}$ and a retract of $\mathcal{G}$ onto G. G is called a deformation retract of $\mathcal{G}$. See [20]. Intuitively, a deformation retraction shrinks a space to a 1-dimensional graph, while keeping the same topology of cycles and connected components.

A. Contour initialization

We first study in a static setting how to construct a contour network that is topologically equivalent to the k-gray band

even with small holes. This algorithm will also be used as a subroutine in the contour repair algorithm, but is confined to the local neighborhood of the contour changes. The contour construction algorithm is motivated by an earlier boundary detection algorithm [10], but is much simplified.

Without loss of generality we assume that $\mathcal{G}$ is connected and has h holes. We compute the *shortest path map* for an arbitrary root, r, which summarizes the shortest path(s) from r to every point in $\mathcal{G}$. The union of points with two or more shortest paths of different homotopy types[1] are called the *cut locus*, C. It is known that C is a forest (let C_i denote the ith tree of C) with m interior vertices (called *SPM vertices*), $h+m$ branches, connecting the h holes and the m vertices to the region boundary. The removal of the cut locus leaves $\mathcal{G}$ simply connected and any shortest path cannot cross a cut branch or go through an SPM vertex [21]. The contour network is computed by putting back shortest paths of different homotopy types, which, together with a part of the cut locus, form the contour network G (Figure 7).

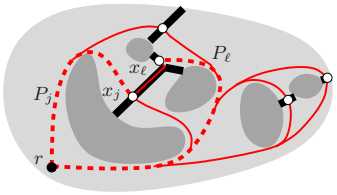


Fig. 7. The contour network G of $\mathcal{G}$. G consists of representative shortest paths of different homotopy types (in red) and the cut locus (in thick black lines). We show two examples of the shortest paths P_j and P_ℓ in dashed curves.

In particular, once we remove the cut locus, the resulting field $\mathcal{G} \setminus C$ has no holes, and its outer boundary consists of the boundaries of holes and segments of cut branches, in an alternating fashion. On each such connected cut segment, we pick a point x_j (the *representative point*) and connect r to x_j along a shortest path (*representative path*). For all representative points on one connected component of the cut locus C_i, their representative paths have different homotopy types – for any two such paths P_j, P_ℓ, as we continuously deform their endpoints x_j to x_ℓ within C_i, one cannot continuous deform P_j to P_ℓ. Notice that there can be multiple representative paths with the same endpoint x_j as long as they have different homotopy types. Let C_i' denote the subtree of C_i that connects the representative points on C_i. Now, the contour network $G = (\cup_j P_j) \cup (\cup_i C_i')$ is the collection of representative paths and the subtrees connecting the representative points in each cut locus component.

1) G is a deformation retract of $\mathcal{G}$: Due to space constraint, we only state the following theorems and defer the proofs in the full version.

Lemma 3.1. *G is a planar graph in $\mathcal{G}$, and each face contains exactly one hole of $\mathcal{G}$.*

Theorem 3.2. *G is a deformation retract of $\mathcal{G}$.*

[1]Two paths with the same start and end points of different homotopy types get around the holes in different ways; thus, one cannot be continuously deformed to the other without jumping over some holes.

2) Implementation in a discrete network: This contour construction algorithm can be implemented in a discrete network as follows. We start from an arbitrary node i and flood the k-gray band. The k-gray band has h holes, which can be detected by discovering the *cut nodes*, i.e., the nodes with two or more shortest paths to i of different homotopy type. Denote by C_i a connected component of the cut nodes. These cut nodes are detected by checking whether two neighboring nodes have their least common ancestor (LCA) "far" away (on the other side of the hole) and their shortest paths "far" apart as well. Such a pair is called a *cut pair*. See Figure 8 for an example. By using appropriate parameters to define "far", holes above a given size can be detected by the recognition of these cut nodes [10].

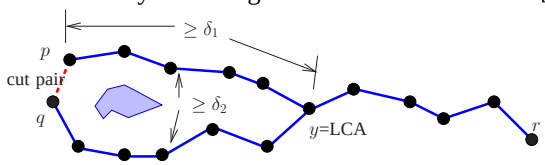


Fig. 8. Definition of a cut pair (p, q).

To find the representative paths, we will first remove the edges between all cut pairs. The cut nodes are left in different connected components. Now we will take one node from each connected component, denoted by x_j, and include in G the shortest path from r to x_j. If any subset of these x_j's belong to the same connected component C_i, they are connected by a tree within C_i. This tree is also included in G.

B. Augmented contour tracking algorithm

We consider the snapshot of two different signal fields. For the first snapshot, we have a topologically valid contour network G for its k-gray region, $\mathcal{G}$. When $\mathcal{G}$ changes to $\mathcal{G}'$, any point that has changed its color (BLACK, WHITE, GRAY) will erase the contour network in its radius-k neighborhood. The part of G unerased contains some broken segments.

The augmented repair algorithm does just one additional operation on top of the repair algorithm in the previous section. In an open neighborhood O_i, recall that the repair node connects through shortest paths to all other bounding segments of O_i. Now, we also include new holes that possibly pop up in O_i. This is by including in addition the "local contour network" in O_i, constructed by the contour initialization algorithm described above.

We argue that after the repair the resulting network G' is topologically equivalent to $\mathcal{G}'$. Network G' has two parts – the old contour network $G_o \subseteq G$ and the newly repaired part G_n.

Theorem 3.3. *The contour network G' is a deformation retract of the k-gray region $\mathcal{G}'$, after the contour repair is done.*

Proof: Consider first the old contour network, G, and the k-gray region $\mathcal{G}$. We remove G from $\mathcal{G}$. We are left with a number of disconnected components, Γ_j. Each one of them is an annulus (band), surrounded by a hole boundary and a face F_i of the old contour network G. By assumption, we have a deformation retract π_j from Γ_j to F_i. Define the *width* of $\mathcal{G}$ as the maximum radius ball centered at a point $p \in \mathcal{G}$ such that the removal of this ball does not change the topology of

$\mathcal{G}$. The width of Γ_j is at most k – otherwise, there will be a BLACK node with GRAY neighbors but do not have RED nodes within distance k. Thus, this node will trigger contour creation. Consider an open neighborhood O_i; it has some bounding

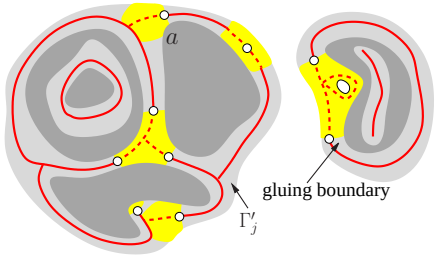


Fig. 9. The repaired network G' and the repair regions (highlighted). Node a has a closed bounding segment and an open bounding segment (adjacent to itself).

segments, some with an open RED end (called *open* bounding segments), some without (called *closed* bounding segments). We define the repair region $R_i \supseteq O_i$ that includes all of the nodes p such that $\pi(p, 1)$ maps to a closed bounding segment of O_i. Intuitively, we are extending the open neighborhood O_i until it hits the closed bounding segments.

Now consider the new contour network G'. We remove the repair regions R_i and the old contour network G_o from $\mathcal{G}'$. This leaves a number of disconnected components Γ'_j, $\Gamma'_j \subseteq \Gamma_j$. If $\Gamma'_j = \Gamma_j$, then we define the deformation retraction π' in Γ'_j to be the same as π. If $\Gamma'_j \subset \Gamma_j$, i.e., part of the contour on F_i has been removed; thus, Γ'_j has the shape of a deformed band. This is because the removal of any point on F_i and its radius k neighborhood will "break" the annulus Γ_j, since the width of Γ_j is at most k. Now Γ'_j is bounded by part of a hole boundary H'_j and part of the face $F'_i \subseteq F_i$, and two "gluing boundaries" adjacent to some repair regions. We map H'_j to F'_j with a continuous function.

We now consider a repair region R_i. We first assume that each open neighborhood O_i is simply connected. Then the repair operation connects with shortest paths from the repair node $r_i \in O_i$ to the bounding segments of O_i. The repaired contour is completely inside R_i. This partitions the repair region into pieces such that each piece is bounded by a contour network segment F'_i, a hole boundary segment H'_i, and gluing boundaries adjacent to some Γ'_j's. We map H'_i to F'_i with a continuous function.

If the open neighborhood has holes, then the repair operation will also include the "local contour network" for O_i, which will partition O_i into disconnected pieces, each face containing exactly one hole with the outer face homotopy equivalent to the outer boundary ∂O_i. This does not interfere with the shortest paths to connect to the bounding segments. Again, the union of any additional shortest paths with the local contour network is still a planar graph.

What this says is that we are able to obtain a continuous mapping of each hole boundary in $\mathcal{G}'$ to a face boundary of G'. With the same argument as in Theorem 3.2, the homotopy equivalence is established. ■

In fact, the proof in the previous theorem states that we can start from a contour network, remove any subset of it, and use the contour repair algorithm to successful repair it. Since we freeze the nodes involved in an open neighborhood under repair, later value changes will be handled after the atomic repair process is finished. The repair operations in different non-overlapping open neighborhoods do not interfere with each other and are handled simultaneously and independently. Thus as long as the signal field stabilizes, the contour network will capture the same topology of the k-gray band. In a dynamic environment, as long as the computation efficiency can catch up with the data change rate, a valid contour network can be maintained. The topological equivalence result implies the following properties listed below with proof omitted from this version.

Corollary 3.4. *The following properties are true when the contour network G stabilizes (i.e., no point switches its state):*

1) *A continuous curve connecting one* BLACK *and one* WHITE *point will have to cross G. A continuous curve connecting two* BLACK *points, from different connected components will have to cross G.*
2) *The contour network is planar with a planar embedding.*
3) *Since all of the repair work happens in the open neighborhood, the communication cost for contour repair is proportional to the amount of contour changes.*

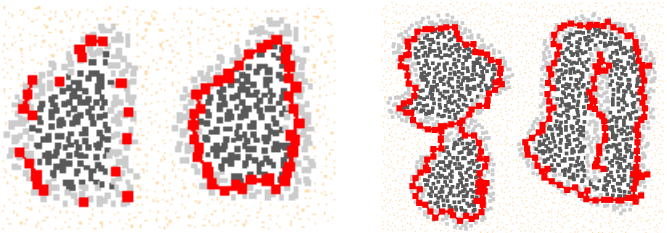
IV. SIMULATIONS

We implemented the contour tracking algorithm described in Section II in a simulator written in C++, since the algorithm covers all cases of contour evolvement, and works well in practice. We simulated on a network with 4000 nodes distributed in a field of size 500×500 units. Each node has transmission radius of 15 units. We set the parameter $k = 2$ by default, and vary k in one set of experiments to discuss its impact on the performance of the algorithm. The simulator takes a data field with arbitrary shape as inputs; in particular, we experimented with both simulated and real data (e.g., Figure14). We simulate dynamic changes among a sequence of stabilized states of a contour field. Between two states, sensor nodes can change their contour values in an arbitrary order. Video clips of some simulated examples can be found at [23].

A. Snapshots of Contours.

We first show a set of snapshots at intermediate stages of the algorithm. Figure 10 shows the process of contour creation when a new black region appears. Initially, a few nodes within the k-gray band elect themselves as leaders and start contour creation. Since contour creation is done in a distributed manner, when new leaders appear, other leaders may already connect to red chains (Figure 10 (i), left). A complete cycle after creation is showed in Figure 10 (i) (right). In some cases, a new black region may be so close to an existing black region that their k-gray bands already merge together. Then, the new contour directly attaches to the existing contour, which guarantees homotopy equivalence (Figure 10(ii)). Contour creation is also triggered when gray/white areas are born inside black regions.

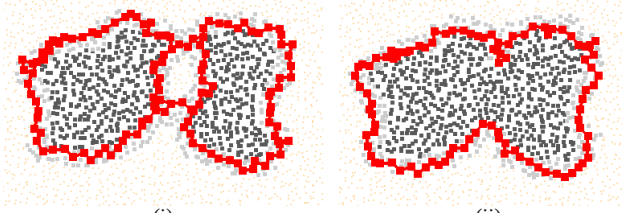
The merging and splitting of contours are symmetric processes. Figure 11 (i) and (ii) show what happens when



(i) (ii)

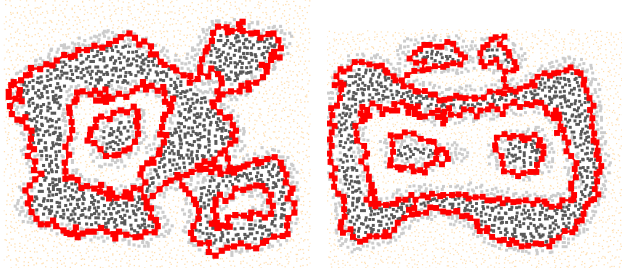
Fig. 10. Contour creation: (i) Left: new leaders appear, and existing leaders connect to red chains. Right: a contour cycle is created. (ii) Left: a new cycle directly attaches to a red cycle nearby. Right: a red chain is constructed when a gray area appears inside a black region.

two originally distant black regions (e.g., the two regions in Figure 10(ii)) move closer and closer. When their k-gray bands just touch, a bridge is automatically constructed to connect those two contours together. In Figure 11 (i), two bridges appear, which exactly capture the white hole between them. If two black regions move towards each other further and eventually merge together, the contours also merge into a larger one (Figure 11(ii)). If we look at these snapshots in a reverse order, they exactly represent a typical process of contour splitting. More interesting snapshots are shown in Figure 12.



(i) (ii)

Fig. 11. Merging and splitting. (i) Two black regions move closer. Their gray bands meet each other and (multiple) "bridges" are built up. (ii) Black regions themselves merge together.



(i) (ii)

Fig. 12. Snapshots of nested contour network.

B. Irregular Network Fields with Holes.

Our contour tracking algorithm is naturally resilient to boundaries and holes with arbitrary shapes. In Figure 13, we show examples of contour networks when a black region attaches to boundaries. The collection of contour pieces correctly separates the black regions from white.

C. Multi-level Contours

Multi-level contours can be easily supported by applying the single-level contour tracking algorithm at each level independently. In Figure 14 (i), we show the multi-level contours corresponding to the elevation data of a small area in Maryland. We take 5 discrete ranges as contour levels of interest. Some sensor nodes may be on multiple contours at the same time if those contours are close to each other. The algorithm correctly maintains the topology of multi-level contours.

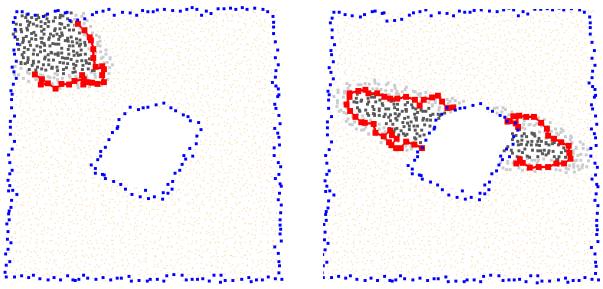
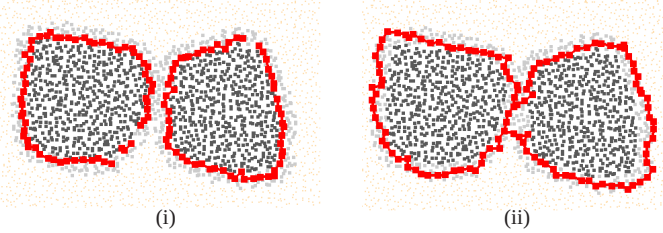


Fig. 13. A contour initially sits at the boundary and successfully passes through a hole in the middle of the network field.

D. Impact of Parameter k.

The parameter k controls the tightness of the contour network to the enclosed black regions. Figure 15 shows different contour networks with different choices of k for the same snapshot of the contour field. When $k = 1$, the k-gray bands are very narrow and have not met each other; thus, those two black regions are still enclosed by two separated red cycles. With $k = 3$, the k-gray bands overlap and red cycles attach to each other. The average distance in terms of hop counts from each red node to a closest black node is about 1.0, 1.4, 1.7, for $k = 1, 2, 3$, respectively; i.e., as k increases, the contour network becomes "loser".

The parameter k also affects the communication cost and the completeness of the contour network. Larger k incurs more transmissions (see Figure 14(ii)); on the other hand, if k is too small, the contour network is easily broken into pieces because a narrow k-gray band may not be a connected piece. Thus, there is a trade-off between communication cost, tightness and completeness. In our simulations, we find that $k = 2$ is suitable for most cases.



(i) (ii)

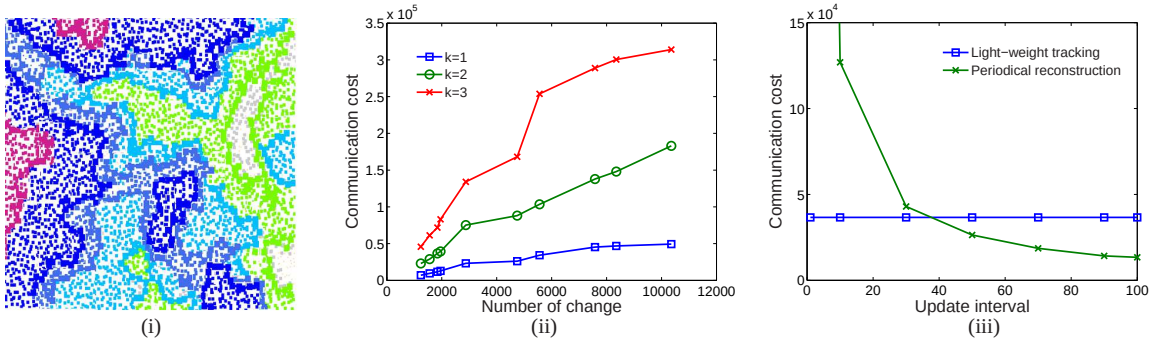
Fig. 15. Tightness of the contour network: (i) $k = 1$. (ii) $k = 3$.

E. Communication Cost.

We evaluate the efficiency of our algorithm in terms of communication cost, measured by the number of transmissions incurred during the transition from one stabilized state to the next. We use 10 different contour fields as inputs, and run simulations on each filed for 5 rounds.

Figure 14 (ii) shows that the communication cost is approximately linear in the number of changes, since all operations in our algorithm are executed locally.

We further compare the performance of our algorithm with a periodic contour reconstruction scheme (see Figure 14 (iii)). We can run any boundary detection algorithm to reconstruct contours periodically. In simulations, we chose to use our contour creation algorithm, since it is essentially a light-weight boundary detection algorithm, which captures the rough



(i) (ii) (iii)

Fig. 14. (i) Multi-level contours on elevation data. Colors represent elevation: purple is the highest and green is the lowest. (ii) Communication cost vs. the number of changes. (iii) Communication cost of periodic construction vs. update interval, compared with our tracking algorithm.

boundaries of the network field, and much cheaper than other accurate boundary detection algorithms. The update interval is defined based on the number of changes and is varied 1 to 100. If we run the periodical reconstruction scheme at every change, it will incur much higher cost, about 30 times the cost of our tracking algorithm, which is out of the range of Figure 14 (iii). With larger intervals, the cost of the periodic reconstruction scheme is reduced, but more critical changes are missed. When the update interval is set to about every 40 changes, it achieves a comparable communication cost with our tracking algorithm, but sacrifices in tracking quality.

V. Conclusions

In this paper we study the problem of contour tracking with binary sensors and propose a light-weight distributed algorithm that locally repairs broken contours as they deform, while guaranteeing that the maintained contours capture the global topological features. We focus on information processing and topology maintenance aspects of the problem. For future work, we plan to explore further the applications of the contour tracking algorithm in processing dynamically changing spatial sensor data. One direction is to combine it with our concurrent work [22] to construct a distributed dynamic contour tree for guided navigation.

Acknowledgement. We would like to thank the support from Stony Brook CEWIT center. X. Zhu, R. Sarkar and J. Gao were partially supported by NSF CAREER Award CNS-0643687. J. Mitchell was partially supported by the National Science Foundation (CCF-0431030, CCF-0528209, CCF-0729019) and by Metron Aviation, under subcontracts from NASA Ames.

References

[1] "http://www.greatduckisland.net/."
[2] R. Szewczyk, A. Mainwaring, J. Polastre, J. Anderson, and D. Culler, "An analysis of a large scale habitat monitoring application," in *SenSys '04: Proc. 2nd Internat. Conf. on Embedded Networked Sensor Systems*, 2004, pp. 214–226.
[3] L. J. Guibas, "Sensing, tracking and reasoning with relations," *IEEE Signal Processing Magazine*, vol. 19, no. 2, pp. 73–85, March 2002.
[4] F. Zhao, J. Shin, and J. Reich, "Information-driven dynamic sensor collaboration," *IEEE Signal Processing Magazine*, vol. 19, no. 2, pp. 61–72, 2002.
[5] J. Aslam, Z. Butler, F. Constantin, V. Crespi, G. Cybenko, and D. Rus, "Tracking a moving object with a binary sensor network," in *SenSys '03: Proc. 1st Internat. Conf. on Embedded Networked Sensor Systems*, 2003, pp. 150–161.
[6] W. Kim, K. Mechitov, J.-Y. Choi, and S. Ham, "On target tracking with binary proximity sensors," in: *Proc. 4th Internat. Symposium on Information Processing in Sensor Networks*, 2005, p. 40.
[7] T. He, S. Krishnamurthy, L. Luo, T. Yan, L. Gu, R. Stoleru, G. Zhou, Q. Cao, P. Vicaire, J. A. Stankovic, T. F. Abdelzaher, J. Hui, and B. Krogh, "Vigilnet: An integrated sensor network system for energy-efficient surveillance," *ACM Trans. Sen. Netw.*, vol. 2, no. 1, pp. 1–38, 2006.
[8] N. Shrivastava, R. M. U. Madhow, and S. Suri, "Target tracking with binary proximity sensors: fundamental limits, minimal descriptions, and algorithms," in *SenSys '06: Proc. 4th Internat. Conf. on Embedded Networked Sensor Systems*, 2006, pp. 251–264.
[9] J. Liu, P. Cheung, L. Guibas, and F. Zhao, "Apply geometric duality to energy efficient non-local phenomenon awareness using sensor networks," *IEEE Wireless Communication Magazine, special issue on Wireless Sensor Networks: Theory and Systems*, 2004.
[10] Y. Wang, J. Gao, and J. S. B. Mitchell, "Boundary recognition in sensor networks by topological methods," in *Proc. of the ACM/IEEE Internat. Conf. on Mobile Computing and Networking*, 2006, pp. 122–133.
[11] S. Funke and C. Klein, "Hole detection or: "How much geometry hides in connectivity?"," in *SCG '06: Proc. 22nd Symposium on Computational Geometry*, 2006, pp. 377–385.
[12] Q. Fang, J. Gao, and L. Guibas, "Locating and bypassing routing holes in sensor networks," in *Mobile Networks and Applications*, vol. 11, 2006, pp. 187–200.
[13] S. Funke, "Topological hole detection in wireless sensor networks and its applications," in *DIALM-POMC '05: Proc. Joint Workshop on Foundations of Mobile Computing*, 2005, pp. 44–53.
[14] S. P. Fekete, A. Kröller, D. Pfisterer, S. Fischer, and C. Buschmann, "Neighborhood-based topology recognition in sensor networks," in *ALGOSENSORS*, Lecture Notes in Computer Science, vol. 3121, 2004, pp. 123–136.
[15] S. P. Fekete, M. Kaufmann, A. Kröller, and N. Lehmann, "A new approach for boundary recognition in geometric sensor networks," in *Proc. 17th Canadian Conf. on Comput. Geometry*, 2005, pp. 82–85.
[16] A. Kröller, S. P. Fekete, D. Pfisterer, and S. Fischer, "Deterministic boundary recognition and topology extraction for large sensor networks," in *Proc. 17th Annual ACM-SIAM Symposium on Discrete Algorithms*, 2006, pp. 1000–1009.
[17] R. Ghrist and A. Muhammad, "Coverage and hole-detection in sensor networks via homology," in *Proc. 4th Internat. Symposium on Information Processing in Sensor Networks*, 2005, pp. 254–260.
[18] J. M. Hellerstein, W. Hong, S. Madden, and K. Stanek, "Beyond average: Toward sophisticated sensing with queries," in *Proc. Information Processing in Sensor Networks*, 2003, pp. 63–79.
[19] S. Gandhi, J. Hershberger, and S. Suri, "Approximate isocontours and spatial summaries for sensor networks," in *Proc. 6th Internat. Symposium on Information Processing in Sensor Networks*, 2007, pp. 400–409.
[20] A. Hatcher, *Algebraic Topology*. Cambridge University Press, 2002.
[21] J. S. B. Mitchell, "A new algorithm for shortest paths among obstacles in the plane," *Ann. Math. Artif. Intell.*, vol. 3, pp. 83–106, 1991.
[22] R. Sarkar, X. Zhu, J. Gao, L. J. Guibas and J. S. B. Mitchell, "Iso-Contour Queries and Gradient Routing with Guaranteed Delivery in Sensor Networks," to appear in *Proc. of the 27th Annual IEEE Conference on Computer Communications (INFOCOM)* , 2008.
[23] Contour tracking videos: "http://www.cs.sunysb.edu/∼xjzhu/contour.html"