Efficient tracking of 2D objects with spatiotemporal properties in wireless sensor networks

Springer

Springer

Efficient tracking of 2D objects with spatiotemporal properties in wireless sensor networks

Guang Jin · Silvia Nittel

Abstract Wireless sensor networks (WSN) are deployed to detect, monitor and track environmental phenomena such as toxic clouds or dense areas of air pollution in an urban environment. Most phenomena are often modeled as 2D objects (e.g., a fire region based on the temperature sensor readings). People model the objects by their properties, and like to know how the properties change over time. This paper presents a distributed algorithm, which uses deformable curves to track the spatiotemporal changes of 2D objects. In order to save the constrained resources in WSN, our distributed algorithm only allows neighboring nodes to exchange messages to maintain the curve structures. In addition, our algorithm can also support tracking of multiple objects. Based on the in-network tracking of deformable 2D curves, we show that many spatiotemporal properties can be extracted by the in-network aggregation. Our experimental results have confirmed that our approach is resource-efficient with regard to the in-network communication and on-board computation.

G. Jin (✉)
Intelligent Automation, Inc., Rockville, MD, USA
e-mail: gjin@i-a-i.com

S. Nittel
Department of Spatial Information and Engineering, University of Maine, Orono, ME, USA
e-mail: nittel@spatial.maine.edu

Fig. 1 Detecting object & boundary by WSN
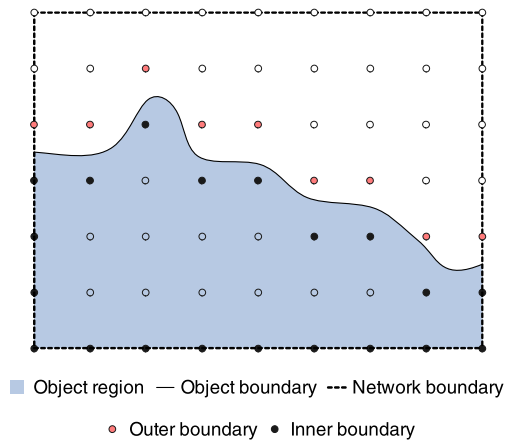
1 Introduction

Wireless sensor networks (WSN) are used to observe environmental phenomena. Many queries about the environmental phenomena are spatiotemporal in nature [29]. For instance, a WSN can be deployed to generate early alarms about wildfires by using heat-resistent sensor nodes [2]. Sensor nodes can collect sensor readings about the wildfires and help us to model the wildfires from different angles. The first one is the field-based model in which a wildfire is a field mapping from the geo-space to an attribute domain [7]. Each sensor node provides real-time readings at the location of node. Based on all sensor readings, a dynamic field map about the underlying phenomena can be generated [13, 32]. The constrained nature of WSN, however, limits the application of field-based models in WSN. From the second angle, users often abstract objects from the phenomenon fields (i.e., as 2D objects with spatiotemporal properties). A 2D object covers a spatial region and changes its shape and location over time. This type of models is known as object-based models. For example, people name wildfires and track them. The abstract spatiotemporal properties, such as the area, location, and topology information, are useful for users to reason about the evolution of 2D objects.

The first step to implement an object-based model in WSNs is to identify objects. This issue is similar to tracking moving objects by WSNs [24], but we cannot attach a hardware identification device, such as RFID [28], to an abstract object. One way to identity a 2D object is to find the object's boundary based on user defined thresholds [30]. Figure 1 illustrates an example. Sensor nodes can exchange sensor readings among neighboring nodes to find local boundary detection results [3, 6, 15, 20]. Since sensor nodes are discretely distributed, local boundary reports represent points around the object boundary, such as the red and black dots in Fig. 1. The point boundary reports can be simplified as the inner boundary reports (e.g., the black dots in Fig. 1) or the outer boundary reports (e.g., the red dots in Fig. 1). The point boundary reports can provide simple snapshots about the object boundary. In the next step, a WSN can link boundary points and return the geometric representation about the 2D objects [25, 34, 37]. Based on the geometric shape (usually polygons), users can define

an object through the shape and location (e.g., the Witch fire from the Witch Creek Canyon or an Island shape [36]).

WSNs work in a dynamic way. They continuously sense and produce information. Therefore, the next logical step which is also the focus of this paper is using a WSN to track particular 2D objects. In this step, a WSN uses geometric shapes to track 2D objects. The tracking shape should be deformable to adapt to the object shape changes. Through tracking 2D objects, a WSN is able to directly provide abstract spatial and spatiotemporal properties about the objects. This paper presents an efficient tracking algorithm for WSNs. We use a closed curve to represent a 2D object, and allow the closed curve to deform and optimize its shape and location to track the 2D object. In addition, our tracking algorithm can also track multiple objects, and adapt the tracking curves to the topology changes caused by the interaction among objects (i.e., splitting and merging). In our algorithm, messages are exchanged locally to maintain the tracking curve. By tracking deformable curves, a network can produce spatiotemporal properties about a 2D object by the aggregated information. In this way, many abstract information and queries can be processed within WSN. The centralized collection of objects' geometric information is not necessary for our approach, whereas most related work simply transmit the geometric information to a central station. Therefore, we expect our approach can save precious resources for WSN. We have implemented our approach in TinyOS [22], and used TOSSIM [22] to evaluate our implementation. The experimental results have confirmed that our algorithm is resource efficient to WSNs.

The remainder of this paper is organized as follows. Section 2 presents the preliminary to abstract 2D objects based on individual sensor readings, and the model of deformable curves. In Sect. 3, we present our revised deformable curve model for the constrained WSN. In Sect. 4, we present a resource efficient way to extend the deformable curve model to track multiple objects. Section 5 provides the detailed tracking algorithm and a discussion on our algorithm. Based on the distributed tracking of deformable curves, Sect. 6 explains how to produce abstract spatiotemporal properties through the aggregated information. Section 7 presents and analyzes our experimental results. We explore the related work in Sect. 8, draw our conclusion and discuss the future work in Sect. 9.

2 Preliminary

This paper mainly focuses on tracking 2D objects. We use a closed curve (or a set of closed curves) to represent the shape of 2D object. This section presents the preliminaries for detecting and tracking 2D objects in WSN.

2.1 Sensor reading, 2D object and boundary

In this paper, we model the world as a 2D space, $\mathbb{R}^2$. s_i is used to identify a sensor node and also its spatial location. We define the immediate neighboring nodes of sensor node, s_i, as a node set,

$$\mathbb{N}(s_i) : \{s_j | s_j \ AND \ s_i \ \text{can directly communicate}\}. \tag{1}$$

Note here, $s_i \in \mathbb{N}(s_i)$, which is defined for our convenience to describe the algorithms in Sect. 3.

Based on the sensor readings, sensor nodes measure an underlying phenomenon. In practice, sensor nodes collect local sensor readings at a user defined sampling rate. The time unit, therefore, is represented as integer values here. A phenomenon is a spatial scalar field that represents the variation of a scalar property over the 2D space [7]. We use $\mathbb{Y}(p, t)$ to indicate the sensor reading at the location p at the time t. $t + 1$ indicates the next time when sensors collect new samples. We use a simple definition for the local object detection results by thresholding sensor readings.

$$\mathbb{O}(p, t) = \begin{cases} 1, & \text{if } \mathbb{Y}(p, t) \geq T; \\ 0, & \text{else.} \end{cases} \tag{2}$$

Based on the given threshold value, T, (2) provides a simple but useful model to derive local object detection results. For example, if $\mathbb{Y}()$ presents the temperature field, we can use $\mathbb{O}()$ based on $T = 200°C$ to define the region of a fire. By exchanging the local sensor readings or object detection results, a sensor node is able to measure the object boundary that separates the object region from the non-object region.

$$\mathbb{B}(s_i, t) = \begin{cases} 1, & \text{if } \mathbb{O}(s_i, t) = 1 \; AND \\ & \exists s_j \in \mathbb{N}(s_i), \, \mathbb{O}(s_j, t) = 0; \\ 1, & \text{if } \mathbb{O}(s_i, t) = 1 \; AND \\ & s_i \text{ is on the network boundary;} \\ 0, & \text{else.} \end{cases} \tag{3}$$

Due to the constrained communication range, the number of immediate neighboring nodes of a sensor node, $|\mathbb{N}(s_i)|$, is usually limited. As defined in (3), $\mathbb{B}()$ is a useful binary function for the local boundary detection. If $|\mathbb{N}(s_i)|$ is large, (3) need be revised to increase the boundary detection quality [30]. If sensors are affected by noise, both (2) and (3) can be extended to a real number function that presents the local boundary or object certainty [3, 6, 15, 20]. In the physical world, the boundary of a 2D object must be closed. A WSN, however, may only cover a subregion of a 2D object. Thus, the network boundary has to be considered as shown by Fig. 1 and (3).

Since sensor nodes are discretely located, boundary reports from (3) are points along the object boundary. Linking boundary points into a closed curve can better assist people to understand the spatial properties about a 2D object. To extract spatiotemporal properties of 2D objects, we need to track 2D objects over time. Next, we need to explain how to track 2D objects by using closed curves.
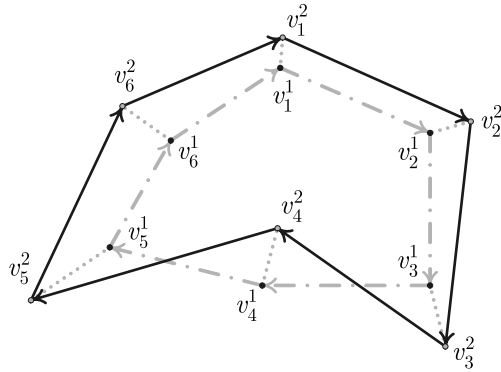
2.2 SNAKE model

We use $\{V^t, E^t\}$ to indicate a closed curve representing the object boundary at time t.

$$V^t = \left\{ v_1^t, v_2^t, \ldots, v_n^t \right\}, \tag{4a}$$

$$E^t = \left\{ \overrightarrow{v_1^t v_2^t}, \overrightarrow{v_2^t v_3^t}, \ldots, \overrightarrow{v_{n-1}^t v_n^t}, \overrightarrow{v_n^t v_1^t} \right\}. \tag{4b}$$

Fig. 2 Example of deformable 2D object tracking

A closed curve consists of n vertices and n edges. The vertices are 2D points (i.e., $v_i^t = (x_i^t, y_i^t)$). The curve is a closed curve, as indicated by (4b). $\{V^t, E^t\}$ is assumed to represent a simple curve (i.e., the curve does not cross itself). The edges in E^t are directed, as illustrated by (4b). We also assume that a WSN can correctly detect the object boundary. In short, compared to the resolution of spatial distribution of sensor nodes, we assume a 2D object needs to be large enough. Due to the monitoring granularity, the boundary of a small 2D object may not be detected. Since the vertices in V^t are sufficient to describe the edges in E^t, as shown by (4b) and (4a), we will use V^t to represent the closed curve in the following parts of this paper. Therefore, we use V^0 to indicate the initial boundary geometry. We allow vertices to "move" in order to adapt the boundary geometry to the underlying object, as illustrated by Fig. 2.

$$leftVertex(v_i^t) = \begin{cases} v_n^t, & \text{if } i = 1, \\ v_{i-1}^t, & \text{else}, \end{cases} \tag{5}$$

$$rightVertex(v_i^t) = \begin{cases} v_1^t, & \text{if } i = n, \\ v_{i+1}^t, & \text{else}. \end{cases} \tag{6}$$

For a vertex v_i^t, we name v_{i+1}^t as the immediate right neighboring vertex of v_i^t; v_{i-1}^t is the immediate left neighbor, by facing the interior region at v_i^t. As shown by (5) and (6), the only exception is that v_1^t is the immediate right vertex of v_n^t; v_n^t is the immediate left vertex of v_1^t. The directed edge connecting v_i^t and $rightVertex(v_i^t)$ is the right edge of v_i^t, while the directed edge from $leftVertex(v_i^t)$ to v_i^t is the left edge of v_i^t. v_i^t and its immediate left and right neighboring vertices form the local angle centered at v_i^t. For $\angle rightVertex(v_i^t)v_i^t leftVertex(v_i^t)$, we define a point p is inside this angle, if p is located on the right of both the right and left edges of v_i^t, as illustrated by Fig. 3.

In the field of computer vision, the deformable curve model is known as the *SNAKE* (or *Active Contour*) model [18]. Here, a deformable curve is used to approximate an object boundary (e.g., coast lines in remotely sensed images) by using sparse points and computing a coarse curve first. Then an energy model is used to adjust the location and number of vertices. In a way, the deformable curve can be treated as a rubber band around a "solid" object. We can use the rubber band to represent the

Fig. 3 Topological relationship
based on local angle
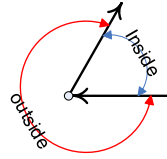
object's boundary. The adjustment of the deformable curve is based on basic physical
rules. When the rubber band is stabilized under different physical forces, the overall
elastic energy is minimal. As shown by Fig. 2, the vertices should be able to "move"
over time under the influence of different "forces", and therefore deform the shape
of the closed curve. At time t, the placement of V^t needs to minimize the "elastic"
energy, E, as,

$$E = \alpha E_{ten} + \beta E_{cur} + \gamma E_{ext}. \tag{7}$$

Equation (7) describes the requirements for a curve to represent an object boundary.
E_{ten} in the first term of (7) is the first order continuity constraint. This term can be
viewed as the tension along the rubber band. If the rubber band is stabilized, the ten-
sion should be equal along the band. In other words, the vertices need to be evenly
distributed along the boundary, which is controlled by E_{ten}. E_{cur} in (7) is the sec-
ond order continuity constraint, and indicates the deformable curve's curvature. E_{cur}
controls the smoothness of deformable curve. E_{ten} and E_{cur} are also called internal
forces, which model the geometric information about the deformable curve. Given
only E_{ten} and E_{cur}, a deformable curve cannot represent a concave shape well. E_{ext},
which is known as the external force or edge strength, provides another force to at-
tach a deformable curve well to a 2D object of arbitrary shape. α, β and γ are relative
weights of each force model, and describe the importance of different forces to the
final shape and location of the deformable curve. By applying the SNAKE model
and using deformable curves to represent 2D objects, distributed sensor nodes are
able to adjust nearby vertices without knowing the global detailed shape of the de-
formable curve. Since the vertex movement is only influenced by different forces, we
need to find appropriate force models, which can be efficiently implemented in the
constrained WSN.

3 In-network tracking of deformable curve

Under the constraints of WSN, sensor nodes should minimize the communication
consumption to maintain the deformable curve structure. In this section, we demon-
strate that our revised SNAKE model achieves this design goal.

3.1 Revised SNAKE model

To use the deformable curve model in WSNs, we constrain that a vertex, v_i^t, can only
move to the location of sensor nodes. We call a sensor node, s_i, a vertex node at
time t, if a vertex v_j^t is at the location of s_i (i.e., $v_j^t = s_i$). To implement the track-
ing algorithm, appropriate force models must be resource friendly. In our proposed

approach, a node locally detects three states, whether the node is located within the object (sensed value above a user threshold), outside an object (value below the user-defined threshold), or on the boundary based on the values of its neighboring nodes.

First, we need to find neighboring boundary nodes, $\mathbb{NB}()$, defined as,

$$\mathbb{NB}(s_i, t) : \{s_j | \; s_j \text{ can communicate with } s_i \text{ directly } AND \\ s_j \text{ detects the object boundary at } t\}. \tag{8}$$

As indicated by (8), $\mathbb{NB}(s_i, t)$ is the set of s_i's neighboring nodes that detect the object boundary at time t. For simplicity, $\mathbb{NB}(s_i, t)$ may contain s_i, if s_i detects the boundary. Sensor nodes are able to prepare local object status and local object boundary status. Sensor nodes only exchange local boundary detection results among immediate neighbors to generate $\mathbb{NB}()$.

External force models for image processing are an active research area. Most available models, however, are expensive to the constrained WSN. For example, gradient vectors can be used as the external force which provides directions towards the 2D object's boundary [35]. Generating the gradient vectors, however, requires several iterations of messages exchanged among sensor nodes (not just among vertex nodes), which is expensive.

In our revised SNAKE-based approach, E_{ext} uses the local boundary detection results provided by $\mathbb{NB}()$. Based merely on $\mathbb{NB}()$, however, an external force may not work well when a vertex node cannot find the boundary report among its immediate neighbors. In the balloon model [4], the proposed E_{ext} contains an outward pressure. By applying the outward pressure, a deformable curve behaves like an inflating balloon to expand itself to represent the object boundary. The balloon model works fine only if the deformable curve is contained within the real object boundary. We need the deformable curve to be able to also "deflate". To better track the object boundary and save the communication cost, we define our external force model as follows.

$$E_{ext}(s_i, t) = \begin{cases} \mathbb{NB}(s_i, t), & \text{if } \mathbb{NB}(s_i, t) \neq \varnothing; \\ \{s_j | s_j \in \mathbb{N}(s_i) \; AND & \text{if } \mathbb{NB}(s_i, t) = \varnothing \\ s_j \text{ is inside the curve}\}, & AND \; \mathbb{O}(s_i, t) = 1; \\ \{s_j | s_j \in \mathbb{N}(s_i) \; AND \; s_j & \text{if } \mathbb{NB}(s_i, t) = \varnothing \\ \text{is not inside the curve}\}, & AND \; \mathbb{O}(s_i, t) = 0. \end{cases} \tag{9}$$

As shown by (9), the proposed external force, E_{ext}, only requires message exchange among neighboring nodes. If some neighboring nodes detect the object boundary, E_{ext} allows the vertex to move onto anyone among them. Note here, a vertex may not need to move, if the vertex node at $t - 1$ detects the boundary at t.

In some situations, a vertex node may lose track of the object boundary (e.g., when a 2D object moves fast), and none of its immediate neighbors detect the object boundary. Its local object detection result and the curve's topology information, however, provide useful information to adapt the curve shape correctly. If a vertex node detects that it is not located within the object and cannot find the object boundary in the neighboring region, the node must be located in the exterior region of the object. In this case, the deformable curve needs to "deflate" locally, as shown by Fig. 4(a). The

Fig. 4 External forces when
$\mathbb{NB}() = \varnothing$
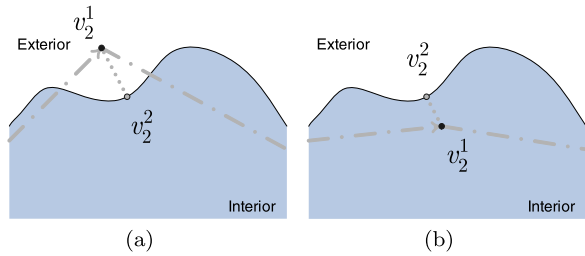
neighboring nodes located in the interior region of the closed curve are the candidate locations for the vertex. As illustrated by Fig. 4(b), if a vertex node detects that it is located inside of the object and finds no boundary in its nearby region, the deformable curve "inflates" locally. The neighboring nodes located in the exterior region of the closed curve are the candidate locations for the vertex. Vertices can eventually find the object boundary by using the proposed E_{ext}. Although our E_{ext} is light-weighted, our model flexibly adapts the deformable curve to track the underlying 2D object.

Equation (9) provides several candidate locations for a vertex node to move to. A vertex can only move to one location among the candidate locations. To calculate the energy weight among the candidate locations, we revise (7) as,

$$E = \alpha E_{ten} + \beta E_{cur}. \tag{7'}$$

Based on (7'), a vertex moves to the location with the minimal energy weight among the candidate locations given by (9).

The internal forces need to be resource efficient as well. A general model for E_{ten} is defined by,

$$\overline{d^{t-1}} - \left| v_i^t - v_{i+1}^t \right|,$$

where $\overline{d^{t-1}}$ indicates the average length of edges,

$$\overline{d^{t-1}} = \frac{1}{n} \sum_{i=1}^{n} n \left| v_i^{t-1} - v_{i+1}^{t-1} \right|.$$

This model requires updating the average edge length, $\overline{d^{t-1}}$, among all vertex nodes if V^t changes. Perrin et al. proposed a new E_{ten} model for detecting object boundaries in digital images [31]. Perrin et al. showed that their E_{ten} model constrains the vertices to be evenly dispersed along the curve. So their E_{ten} model is ideal for our tracking quality requirements. We slightly modify their E_{ten} model. Our E_{ten} model is resource-efficient and only requires message exchange among consecutive vertex nodes, as defined by,

$$E_{ten} = Var\left(\left| v_i^{t+1} - v_{i-1}^t \right|, \left| v_i^{t+1} - v_{i+1}^t \right| \right). \tag{10}$$

For a candidate location, v_i^{t+1}, of v_i^t, $Var()$ measures the variance of the lengths of two consecutive edges, $|v_i^{t+1} - v_{i-1}^t|$ and $|v_i^{t+1} - v_{i+1}^t|$. When the two edges are equal length, E_{ten} is zero. To minimize E_{ten}, the vertices need to be located at equal intervals along the curve.

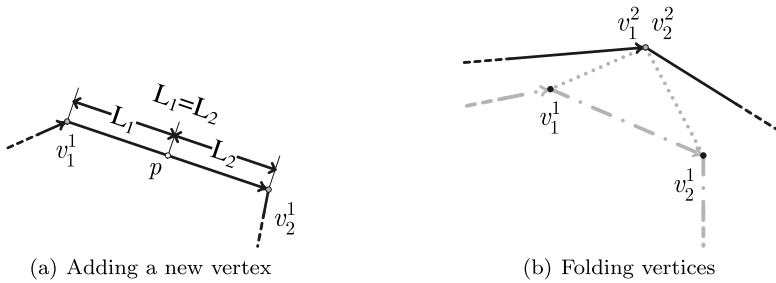(a) Adding a new vertex   (b) Folding vertices

Fig. 5 Examples of dynamic adding and folding

When the 2D object expands and shrinks, the deformable curve, like the rubber band, should expand and shrink simultaneously. The parameter D_{split} controls the number of vertices when the curve deforms.

$$\begin{cases} |v_{i+1}^t - v_i^t| \leq D_{split}, & \text{No change;} \\ |v_{i+1}^t - v_i^t| > D_{split}, & \text{Add a vertex between} \\ & v_i^t \text{ and } v_{i+1}^t. \end{cases} \tag{11}$$

When the distance between v_i^t and v_{i+1}^t is larger than D_{split}, a new vertex is added between v_i^t and v_{i+1}^t. As illustrated by Fig. 5(a), to ensure the even vertex spacing, the new vertex is placed at

$$\left(\frac{x_i^t + x_{i+1}^t}{2}, \frac{y_i^t + y_{i+1}^t}{2} \right). \tag{12}$$

D_{split} ensures the largest disparity in the vertex spacing, and influences the tracking quality of the deformable curve. When the deformable curve shrinks, multiple vertices may move to a single sensor node. Some vertices moving onto a single node are consecutive neighbors, and can be folded into a single vertex, as shown by Fig. 5(b). A more complex case will be explained in Sect. 4.

E_{cur} controls the curve's smoothness. We use the value of the inner angle to represent E_{cur}. The second order curvature can be used to represent the smoothness, which minimizes the angle variation of three consecutive angles [31]. In short, the second order curvature model requires that the three consecutive angles are similar. To find the curvature value of the next location, p, for v_3^1 in Fig. 6(b), the second order curvature model needs to know the value of three internal angles, $\angle p v_2^1 v_1^1$, $\angle v_4^1 p v_2^1$ and $\angle v_5^1 v_4^1 p$. The second order curvature needs a vertex location to be updated among five consecutive vertices, which is expensive in communication. In Fig. 6(b), the updated location of v_3^1 should be sent to v_1^1, v_2^1, v_4^1 and v_5^1. To save the energy and communication cost, we choose the first order curvature defined as,

$$E_{cur} = Var\left(\pi, \angle v_{i+1}^t v_i^{t+1} v_{i-1}^t \right). \tag{13}$$

As indicated by (13) and illustrated by Fig. 6(a), the first order curvature model is biased towards straight lines. The first order curvature requires a vertex update to be
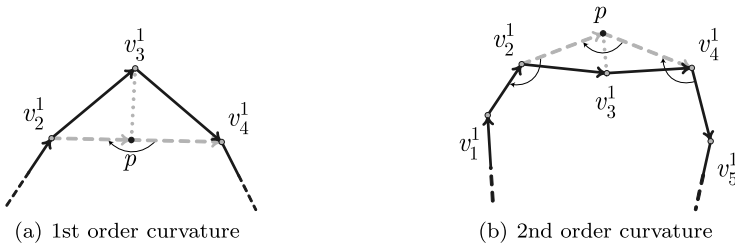
Fig. 6 Curvature models

exchanged only among three consecutive vertex nodes. For example, in Fig. 6(a), the updated location of v_3^1 should only be sent to v_2^1 and v_4^1. Our experiments showed that the first and second order curvature models have almost the same tracking quality. One possible explanation is that E_{cur} dominates the final curve shape only when D_{split} is large. Similarly along a rubber band, a local bend only affects a nearby area.

The revised E_{ext}, E_{ten} and E_{cur} models are light-weighted, and need message exchange among neighboring nodes only. Based on the revised E_{ext}, E_{ten} and E_{cur} models, a WSN efficiently "moves" vertices and therefore tracks the underlying 2D objects. We also need to consider the topology changes when multiple 2D objects interact, which will be explained by the next section.

4 Tracking of multiple objects

When multiple 2D objects change their shapes and locations in space, basically two types of topological changes (i.e., splitting and merging) are involved [14]. Deformable curves representing 2D objects consequently should adapt their shapes to the topological changes. The original SNAKE model is too rigid to do so, since the connected edges are unbreakable. A flexible model is necessary for deformable curves to adapt to the topological changes. Note here, we do not consider the dimensional changes in this paper. In other words, we assume that the remaining objects after the splitting or merging changes need to be large enough for a WSN to correctly detect the boundary and treat them as 2D objects.

4.1 Breaking and reconnecting edges

Today, several revised SNAKE models have been proposed to use breakable curves to track 2D objects [21, 27]. Most models are based on a centralized infrastructure, which is not suitable for the constrained WSN. In the T-Snake approach [27], the space is partitioned into non-overlapping triangles. In a triangle cell, nonconsecutive edges need to be removed and replaced by a single edge. The T-Snake approach, however, faces the ambiguity caused by different triangulation patterns. In Fig. 7, the solid lines indicate the edge of deformable curves; the dotted lines represent the triangulation partition. The edges in Fig. 7(a) are identical to the edges in Fig. 7(b). Due to the different triangulation patterns, the edges in Fig. 7(a) need to be removed, whereas the same edges in Fig. 7(a) can be kept. A global uniform triangulation

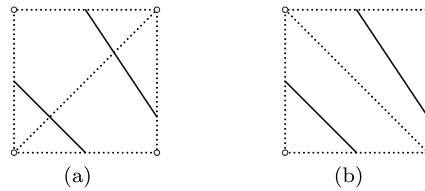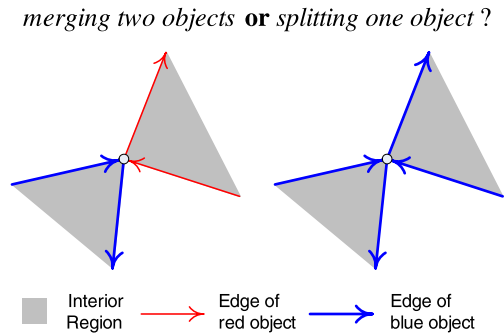**Fig. 7** Ambiguity caused by different triangulation patterns

(a)   (b)

Fig. 8 Ambiguity when two 2D objects touches at a single point

merging two objects **or** *splitting one object* ?

Interior Region   Edge of red object   Edge of blue object

pattern is necessary for the T-Snake approach. Finding the global triangulation pattern in the constrained WSN, however, has to consume additional resources, especially if sensor nodes are unevenly distributed or nodes are mobile [12, 32].

One observation is that any 2D object can be represented by a set of simple closed curves. A 2D object can contain holes. A hole can also be represented by a simple closed curve. Nonconsecutive edges in a simple closed curve cannot intersect, overlap or touch with each other. Purely based on the geometric shape of deformable curves, our model focuses on converting non-simple curves into simple curves.

Two 2D objects can touch at a single point. If we try to reconnect the edges linked to the same point, we shall face an ambiguity. The reconnected edges simultaneously can indicate a 2D object is splitting, as illustrated by Fig. 8. To better adapt the deformable curves to the topological changes, our model is based on the detection and removal of overlapping and intersecting edges.

The original SNAKE model is based on physical laws. It is intuitive to explain our model by the example of soap bubbles. When two soap bubbles are merging, some parts of the bubble walls from two bubbles overlap first. Then the overlapping bubble walls break, and two bubbles become a single bubble. Figure 9 shows a zoom-in picture of Fig. 10(b). Let us consider one end-point of the overlapping edges in Fig. 9. The end-point is actually covered by two different vertices that were previously located at different points but moved onto the same point. By removing the overlapping edges, we get two open curves. One of the vertices on the same point then has the right edge removed; another one has the left edge removed. The two vertices are locally reconnected and merge into a single vertex. The merged vertex now has the left and right edges from remaining edges of the previous two vertices. In this way, two open curves are reconnected into a single closed curve.

When a bubble is splitting, a part of the bubble wall overlaps another part from the same bubble, as illustrated by Fig. 10(a). This can also be represented by Fig. 9.

Fig. 9 Removing and reconnecting edges
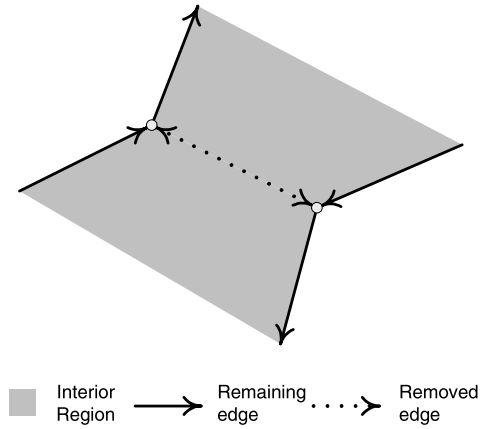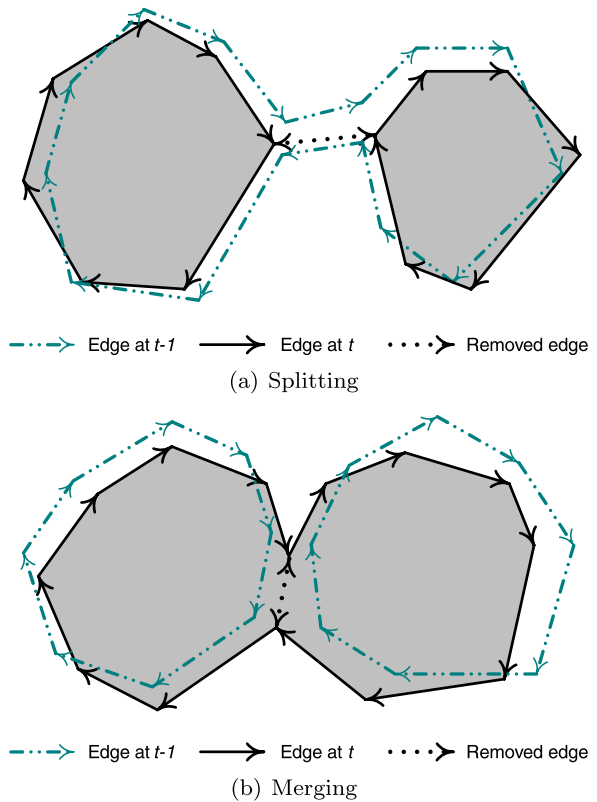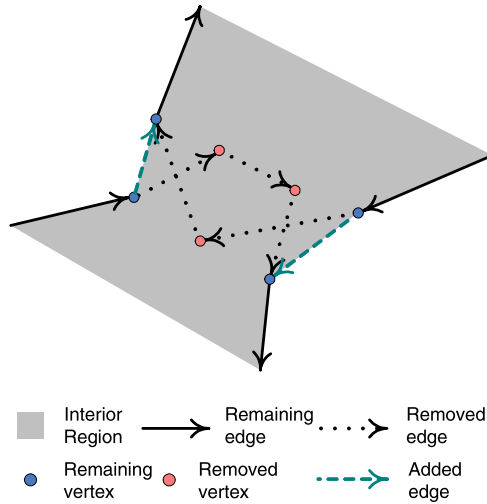

Fig. 10 Example of splitting and merging

(a) Splitting

(b) Merging

The only difference is that the edge direction is reversed, and the interior and exterior regions are reversed. An interesting observation from Fig. 9 is that the detection of overlapping edges can be done locally on distributed nodes.

Fig. 11 Removing and
reconnecting intersecting edges

Due to the discrete distribution of sensor nodes, the vertex movement cannot be continuous. In some cases (e.g., uneven node distribution), edges may intersect with each other. Since the closed curves are simple, the intersecting edges need to be removed. We get open curves after removing the intersecting edges. As explained by Fig. 11, those vertices located on the non-boundary region should be removed, when the edge intersection occurs. For a pair of removed intersecting edges, two vertices (one without left neighbor, another one without right neighbor) may remain. A new edge, therefore, should be added here to reconnect the open curves, as shown by Fig. 11. The two vertices on the open curves consequently are consecutive vertex neighbors. As explained above, D_{split} is the longest edge length. Suppose R_{comm} indicates the communication range of wireless radio. Due to the broadcasting nature of wireless channel, if $D_{split} \leq \sqrt{2} R_{comm}$, no additional communication is required to detect the intersecting edges based on our in-network deformable curve tracking.

5 Algorithms

In the proposed approach, vertex location information is exchanged among neighboring vertex nodes. When a vertex v_i^t is located at a particular sensor node, the sensor node needs to know the locations of v_{i-1}^t and v_{i+1}^t. By assuming that the vertices are facing the exterior region, we use the "LEFT" and "RIGHT" relations to identify the neighboring vertices. For the vertex node of v_i^t, we use two local variables, *leftVertex* and *rightVertex*, to store its left and right neighboring vertices v_{i-1}^t and v_{i+1}^t. A timer is used in our implementation to control the sensors and the vertex movement. When time elapses from t to $t + 1$, sensors collect new local readings. Afterwards, sensor nodes exchange local object and boundary detection results. We use GPSR [17] as the communication protocol, and assume a position service running on the background [5, 23]. Sensor nodes therefore communicate with each other based on their locations.

We also assume that the background services handle node failures and communication failures.

5.1 Pseudo-codes and description

Based on the neighboring boundary detection at time $t + 1$, a vertex node uses the location of previous neighboring vertices at time t to calculate the vertex's next location, as shown by Algorithm 1.

After exchanging local boundary detection results, a vertex node finds nearby nodes, which detect the object boundary. If a neighboring node detects the boundary, the node is a candidate for the vertex's next location as shown by Algorithm 1. If the vertex node cannot detect the boundary in the nearby area, the node uses the local angle's topology information and its local object detection result to find the next candidate location. If a vertex node detects neither the object nor the object boundary in the nearby area, the candidate locations are within the interior region defined by the local angle. If a vertex node detects the object but does not find the object boundary in the nearby area, the candidate locations are within the exterior region, as illustrated by Algorithm 1. Among the candidate locations, the location with the minimal tension and curvature energy is the next location for the vertex. A designation message is sent to the sensor node located at the next location. When a sensor node receives a vertex movement message, the sensor node caches the vertex movement

---

Algorithm 1 Finding the next location of v_i^t

Require: Sensor nodes exchange local object detection, *objectDetected*, and boundary detection results among immediate neighboring nodes, *NSensors*. The neighboring boundary reports are stored into a point array *NBReports*.

Ensure: v_i^t moves to a node at location v_i^{t+1} with minimal energy.

```
 1: if NBReports.length ≠ 0 then
 2:     CandLocs ⇐ NBReports
 3: else
 4:     for all s ∈ NSensors do
 5:         if objectDetected = FALSE then
 6:             if s INSIDE ∠v_{i+1}^t v_i^t v_{i-1}^t then
 7:                 CandLocs.add(s)
 8:             end if
 9:         else
10:             if s OUTSIDE ∠v_{i+1}^t v_i^t v_{i-1}^t then
11:                 CandLocs.add(s)
12:             end if
13:         end if
14:     end for
15: end if
16: MinE ⇐ +∞
17: r ⇐ v_{i+1}^t
18: l ⇐ v_{i-1}^t
19: for all s ∈ CandLocs do
20:     E_{ten} ⇐ Var(|s − l|, |s − r|)
21:     E_{cur} ⇐ Var(π, ∠rsl)
22:     E ⇐ αE_{ten} + βE_{ten}
23:     if E < MinE then
24:         MinE ⇐ E
25:         nextLoc ⇐ s
26:     end if
27: end for
28: return nextLoc
```

---

into a cached vertex movement array, *CVM*. An element in *CVM* contains the vertex's previous location, and the vertex's previous left and right neighboring vertices. Since multiple vertices may move onto a single node at the same time, Algorithm 2 is used to fold multiple vertices.

Multiple vertices may move onto the same sensor node. After receiving a vertex movement message, a sensor node caches the vertex movement into a vertex movement array, *VM*. An element in *VM* contains the vertex's previous location, and the vertex's previous left and right neighboring vertices. Some vertices can be folded into a single vertices (e.g., consecutive left and right neighboring vertices), as illustrated by Algorithm 2. By comparing the *LEFT* and *RIGHT* relationships among vertices, Algorithm 2 folds consecutive vertices into a single vertex and insert the vertex into the vertex list, *VL*. If vertices are not consecutive (e.g., the vertices are from two 2D objects), *VL* may contain multiple vertices. Till now, a vertex movement is finished. The new vertex node then notifies the vertex's current location to its previous left and right vertex nodes. The current left and right vertex nodes may get the message through the previous left and right vertex nodes. The updated location messages are exchanged only among neighboring vertices through necessary relays. After receiving the updated location of its neighboring vertices, a vertex node knows the locations about its current right and left vertices. After the vertex updates are done, a vertex

---

Algorithm 2 Folding consecutive vertices

Require: A sensor node receives multiple vertex movement messages and caches the messages into cached vertex movement array *VM*.
Ensure: Folding vertices on the local sensor nodes and prepare the vertices list *VL*.
1:   $VL \Leftarrow VL.init()$
2:   **repeat**
3:       $mostLeft \Leftarrow VM.getFirst()$
4:       $VM \Leftarrow VM.remove(mostLeft)$
5:       **repeat**
6:           $toRepeat \Leftarrow FALSE$
7:           **for all** $m \in VM$ **do**
8:               **if** $mostLeft.preLeft = m.preLocation$ **then**
9:                   $VM \Leftarrow VM.insert(mostLeft)$
10:                  $mostLeft \Leftarrow m$
11:                  $VM \Leftarrow VM.remove(m)$
12:                  $toRepeat = TRUE$
13:              **end if**
14:          **end for**
15:       **until** $toRepeat = FALSE$
16:       $mostRight \Leftarrow mostLeft$
17:       **repeat**
18:           $toRepeat \Leftarrow FALSE$
19:           **for all** $m \in VM$ **do**
20:               **if** $mostRight.preRight = m.preLocation$ **then**
21:                   $mostRight \Leftarrow m$
22:                   $VM \Leftarrow VM.remove(m)$
23:                   $toRepeat = TRUE$
24:              **end if**
25:          **end for**
26:       **until** $toRepeat = FALSE$
27:       $v \Leftarrow newVertex()$
28:       $v.preRight \Leftarrow mostRight.preRight$
29:       $v.currentRight \Leftarrow null$
30:       $v.preLeft \Leftarrow mostLeft.preLeft$
31:       $v.currentLeft \Leftarrow null$
32:       $VL \Leftarrow VL.insert(v)$
33:   **until** $VM.length = 0$
34:   return VL

Algorithm 3 Adding a vertex

Require: Vertex nodes update their current location to the left and right neighbors.
Ensure: Adding a new vertex operation if the distance between a vertex and its right vertex is larger than D_{split}.

1: **if** $|myLocation - rightVertex| > D_{split}$ **then**
2:      $v.x \Leftarrow \frac{myLocation.x + rightVertex.x}{2}$
3:      $v.y \Leftarrow \frac{myLocation.y + rightVertex.y}{2}$
4:      $newVertex \Leftarrow PositionService.FindNearestNode(v)$
5:      $notifyNewVertexTo(rightVertex)$
6:      $designateNewVertex(newVertex)$
7:      $rightVertex \Leftarrow newVertex$
8: **end if**

---

node checks the distance to its right vertex. If the distance is larger than D_{split}, a new vertex is added in between, as illustrated by Algorithm 3.

The new vertex is the middle point of the local vertex and its right neighboring vertex. Since sensor nodes are discretely distributed, the nearest sensor node to the middle point is found through the background position service [5]. As shown by Algorithm 3, if a new vertex is inserted, the nearby vertex links are updated, and the new vertex node is notified.

After receiving the updates from neighboring vertices, a sensor node needs to update the corresponding vertex entry in the vertex list, *VL*. An element in *VL*, therefore, contains the locations of the vertex's current right and left neighboring vertices. Based on the content of *VL*, a sensor node detects the overlapping edges locally. After removing overlapping edges, the open curves need to be reconnected. Some vertices may also be removed accordingly as illustrated by Algorithm 4. If no vertex is remained (e.g., a vertex has its current right and left neighboring vertices overlapping), the sensor node becomes a non-vertex node. The edge between the local sensor node and *brokenNeighbor* indicates the removed overlapping edges. The location of *brokenNeighbor* is useful to determine whether the topological change is *splitting* or *merging*. After removing the overlapping edges and reconnecting the open curves, the remained vertex moves based on the force models afterwards.

In the in-network deformable curve tracking, vertex nodes need to update the current vertex locations to neighbors. Vertex nodes can detect the intersecting edges based on the broadcasting vertex location updates. No additional communication is required over the deformable curve tracking, if $D_{split} \leq \sqrt{2}R_{comm}$. After the pair of intersecting edges, *IE*, are detected, the four vertex nodes need to be notified. Some vertices may need to be removed if the vertices are not located on the object boundary, as illustrated by Algorithm 5. Intersecting edges need to be removed. We need to re-close the open curves by reconnecting the vertices, as explained by Algorithm 5. Similar to the *brokenNeighbor* in Algorithm 4, the location of intersecting edges in Algorithm 5 also helps to determine the type of this topological change.

5.2 Discussion

We assume the initial curve V^0 is given. The initial curve V^0 can be found by distributed algorithms [11, 25, 34], or from the distributed detection result based on the different models [36]. For example, the emerging of a 2D object matching a user-defined shape can provide the initial boundary V^0. Due to the constrained environment, the

---

Algorithm 4 Removing overlapping edges and reconnecting open curves

---

Require: A sensor node folds multiple vertices and has the current neighboring vertices' locations updated into the vertices list *VL*.
Ensure: Removal of overlapping edges and corresponding vertices; reconnecting open curves; reporting the removed edge.

```
 1: for all v ∈ VL do
 2:     for all o ∈ VL do
 3:         if v.currentLeft = o.currentRight then
 4:             brokenNeighbor ⇐ v.currentLeft
 5:             v.currentLeft ⇐ null
 6:             o.currentRight ⇐ null
 7:         end if
 8:     end for
 9: end for
10: for all v ∈ VL do
11:     if v.currentLeft = null AND v.currentRight = null then
12:         VL.remove(v)
13:     end if
14: end for
15: for all v ∈ VL do
16:     for all o ∈ VL do
17:         if v.currentLeft = null AND o.currentRight = null then
18:             v.currentLeft ⇐ o.currentLeft
19:             VL.remove(o)
20:         end if
21:     end for
22: end for
23: return brokenNeighbor
```

Algorithm 5 Removing intersecting edges and reconnecting open curves

---

Require: The intersecting edges have been detected; an *IE* structure contains the four vertices of the intersecting edges; consequently, four vertices have been notified about the intersection and run this algorithm.
Ensure: Removal of intersecting edges and corresponding vertices; reconnecting open curves; reporting the removed edges and vertices.

```
 1: if localBoundaryStatus = false then
 2:     resignVertex(mySelf)
 3:     return reportRemovedVertex(myLocation)
 4: end if
 5: for i = 0 to 1 do
 6:     if i = 0 then
 7:         j ⇐ 1
 8:     else
 9:         j ⇐ 0
10:     end if
11:     if myLocation = e[i].leftVertex then
12:         rightVertex ⇐ e[j].rightVertex
13:         return reportRemovedEdge(e[i])
14:     else if myLocation = e[i].rightVertex then
15:         leftVertex ⇐ e[j].leftVertex
16:         return reportRemovedEdge(e[i])
17:     end if
18: end for
```

V^0 shape given by a distributed object detection is usually coarse, such as a simple rectangle [36]. Our tracking algorithm changes and optimizes the shape and location of V^0 based on the revised SNAKE model to well attach to the 2D object. In related approaches, the geometric boundary is collected and reported back to the base station periodically. Our tracking algorithm, on the other hand, is able to just report the incremental changes of object boundary. Particularly, a base station is able to track the object by using the vertex changes (i.e., the movement, addition and removal of vertex). The vertex change can also be extended to the spatiotemporal range change.

In other words, a vertex change is reported back, only if the vertex has moved out of a predefined range. Furthermore, the tracking of deformable curves can easily be extended to support the in-network extrapolation of the curves' future location and shape. Due to the space limitation, we will leave on these extensions to our future work.

The proposed tracking algorithm for deformable curves maintains the curves by localized message exchange. When a vertex moves, the vertex node sends a designation message to one of its immediate neighbors, and resigns. The new vertex node reports the updated vertex location to the previous left and right vertex nodes. The previous left and right vertex nodes may need to relay the update messages to the vertices' current locations. D_{split} roughly bounds the geographical range for a vertex update message to be transmitted. If the curve keeps a constant number of vertices, the maintenance cost of the tracking algorithm is constant. If the curve expands and requires more vertices, the maintenance cost increases linearly to the number of vertices.

D_{split} also controls the number of vertices along a closed curve. If D_{split} is large, fewer vertices are added when a curve deforms. D_{split} is useful to control the quality of the deformable curve to represent the underlying 2D object. Similar techniques have also been applied to simplify the curve shape [11]. Compared with reporting points along the object boundary, the network requires less communication to send linked vertices, if D_{split} is large. The difference is approximately scaled by D_{split}, since only the two end vertices of a line with $length = D_{split}$ represents the whole set of points along the line.

The location of *brokenNeighbor* is useful to locally judge the type of the topological change, as explained by Algorithm 4. After removing overlapping edges, a sensor node forms a new angle, which has the remaining left and right edges as the new angle's left and right edges. By comparing Fig. 10(b) and Fig. 10(a), we shall see that if *brokenNeighbor* is within the new angle, then the topological change is a merging event. If *brokenNeighbor* is outside the angle, a splitting event occurs. In some cases, a 2D object may partially merge itself. For example, a band is bent into a ring. Similarly, a ring can be broken into a band. To better solve this issue on how to efficiently and locally determine the type of the topological change, we may need to assign unique identifications to 2D objects [9]. For example, a sensor node can combine the object ID of the removed edge with the topological test result based on the location of *brokenNeighbor* to determine if a ring is newly formed. We do not address this issue in detail, since it is beyond the scope of this paper.

As illustrated by Fig. 9, Algorithm 4 requires no additional communication cost over the in-network deformable curve tracking. The detection of intersecting edges may need additional communication cost. If D_{split} is small enough, vertex nodes are able to detect intersecting edges through the broadcasting vertex location updates. After intersecting edges are found, the four vertex nodes need to be notified. Algorithm 5 removes the intersecting edges and reconnects the representative curves, as shown by Fig. 11.

The proposed tracking algorithm does not require a particular network layout, such as the grid layout. As illustrated by Algorithm 1, a vertex node can pick one neighbor's location as the vertex's successive location. The local boundary detection result

and the location of a neighbor, on the other hand, determine the "elastic energy", if a vertex moves to the neighboring node. A vertex node only chooses one neighbor as the successive location.

A message may get lost during the wireless transmission. The acknowledgement and retransmission are necessary. This issue, however, is beyond the scope of this paper. We assume that the underlying geo-routing protocols assist the proposed tracking algorithm to handle communication failures [17, 19]. The death of non-vertex nodes only affects the number of neighboring location points where a vertex may move to. The proposed tracking algorithm can still run smoothly, if non-vertex nodes die. A more serious problem may occur, if a vertex node dies before it correctly resigns and notifies the resignation to its neighboring vertex nodes. A live vertex node can periodically notify its neighboring vertex nodes about its current status that may include its health and location. If a vertex node cannot hear the status message from its neighboring vertex nodes, the node can generate a vertex node failure message and pass the message along the deformable curve. The deformable curve, consequently, can be terminated. A distributed boundary geometry formation algorithm may be called again to regenerate the representative curve [11, 25, 34]. Our tracking algorithm can be called again afterwards. The proposed tracking algorithm also requires a synchronization service to synchronize nodes [8]. Due to the space limitation, however, we cannot discuss these issues in detail.

Based on the algorithms described in this section, a WSN is able to track 2D objects separately and their interactions in network. A WSN can update the deformable curves to users and let users to get the spatiotemporal properties from the geometric information. The deformable curve tracking algorithm provides more than just the snapshot results about representative curves. Based on the deformable curves, a WSN is able to directly extract abstract spatiotemporal properties of 2D objects without returning users the detailed geometric information about the representative curves.

6 Aggregated abstract information

As explained by A. Galton, several abstract spatiotemporal properties are useful for cognition, linguistics and reasoning [10]. In daily life, people can describe and exchange information about 2D objects by abstract spatial and spatiotemporal information without any graphical aid. In Sect. 4, we have explained how to adapt the representative curves to the topological changes involved by the interaction between multiple objects. In [16], we have shown how to use the aggregation over deformable curves to extract the several spatial and spatiotemporal properties about 2D objects. By using these aggregation operations, an SDMS is able to answer many useful spatial and spatiotemporal queries without reporting the detailed geometric representation about 2D objects. Therefore, communication resources can be saved.

For instance, the area of a 2D object can be aggregated based on (14).

$$A^t = \frac{1}{2} \sum_{i=1}^{n} \left(x_i^t y_{i+1}^t - x_{i+1}^t y_i^t \right). \tag{14}$$

In this aggregation, a vertex node prepares its local partial results based on its location and its right neighboring vertex. The area about the region covered by a 2D object curve can be computed through the in-network aggregation [26].

$$A^{t+1} - A^t = \sum_{i=1}^{n} \left(DA1_i^{t+1} + DA2_i^{t+1} \right), \tag{15}$$

where

$$DA1_i^{t+1} = \frac{1}{2} \left| v_i^{t+1} - v_i^t \right| \left| v_{i+1}^t - v_i^t \right| \sin \angle v_i^{t+1} v_i^t v_{i+1}^t, \tag{16a}$$

$$DA2_i^{t+1} = \frac{1}{2} \left| v_{i+1}^{t+1} - v_{i+1}^t \right| \left| v_{i+1}^{t+1} - v_i^{t+1} \right| \sin \angle v_{i+1}^t v_{i+1}^{t+1} v_i^{t+1}. \tag{16b}$$

The in-network aggregation has been shown to be an efficient paradigm to process sensor data in wireless networks. To further improve the processing efficiency, the report suppression can be used [33]. We find that many real-time queries about abstract properties can be answered by using the temporal changes in these properties. For example, the area change as indicated by (15) can be used to answer area queries. A local vertex node prepares the local area change based on the nearby vertices' locations. The local area change values, $DA1_i^{t+1}$ and $DA2_i^{t+1}$, are signed scalars, as indicated by (16a) and (16b). By using (15), a vertex node is able to suppress local partial aggregation reports. For example, if the local area change is zero, a vertex node can suppress the local report to its parent node.

The in-network aggregation approach can easily be extended to other spatial properties and the corresponding spatiotemporal property changes, such as *Minimal Bounding Rectangle* (MBR), perimeter, centroid. In next section, the experimental results will show the efficiency of our approach.

7 Experimental evaluation

We implemented our distributed deformable curve tracking algorithms in TinyOS [22], and used CLDP [19] which is an enhanced TinyOS implementation of GPSR as the routing protocol. We run our codes in TOSSIM [22], and set the simulated environment as follows. The network was in a grid layout. In the network, 169 sensor nodes were distributed evenly in a 100×100 2D space at the interval of 8. The root node was located at $(2, 2)$, and connected to a base station. The wireless radio range was 10, which allowed a sensor node to directly communicate with up to four neighbors within the range. The weights α and β were equal to 1. Sensor nodes collected sensor readings based on four 100×100 video clips. Each sensor node collected sensor readings from the corresponding pixel values in the video clips based on the node's location. The video clips simulated both simple and complex 2D objects, including convex objects, concave objects and objects with hole. The communication cost is based on the number of packets including the packets used for multi-hop routing. We ran each test multiple times and measured the average cost to compare different approaches. Table 1 summarizes the parameter settings in our experiments.

Table 1 Parameter settings

Parameter	Value	Parameter	Value
Network Layout	Grid	Network size	169
Node Interval	8	Radio Range	10
α	1	β	1
Root location	(2, 2)		

In the first set of tests, we focus on tracking single simple 2D object. To control the curve tracking quality, D_{split} was set to 18. Two video clips containing two different objects were used. The initial shapes of both objects were a solid circle with *radius* = 25. The initial curves were both an inscribed regular octagon of the circle. The object 1 started with *center* = (35, 35), and moved $x + 4, y + 4$ in each frame while kept the size constant. The object 2 started with *center* = (50, 50), and enlarged *radius* + 4 in each frame while kept the center unchanged. A video frame was updated to TOSSIM in every 700 seconds. Sensor nodes were woken up in every 350 seconds to collect updated sensor readings, detect objects and boundaries, and deform the tracking curves. A sensor node took a sensor reading as the corresponding pixel value in the concurrent video frame based on the node's location.

We implemented the first order and second order curvature models in our experiments. The two curvature models preformed almost same in the tracking quality, since we set D_{split} to a small value. The first order curvature model prefers the local angle to be π, while the second order curvature model constrains local three consecutive angles to be similar. The first order curvature model requires the location update of a vertex to be exchanged among three neighboring vertex nodes. The second order model needs to exchange a vertex location update message among five consecutive vertex nodes to update the three angles' values. The second order model also needs more communication resources to send the folding vertices and adding vertex notifications. Figure 12 shows the average maintenance costs of the first and second curvature models from our tests. The first order curvature model consumes as around 36% maintenance communication cost as required by the second order curvature model. Since the first order curvature model requires less maintenance cost and shows no difference in the tracking quality, we did the following tests only based on the first order curvature model.

We compared the communication cost of tracking deformable curves against the cost of reporting boundary points. Figure 13 illustrates the average communication costs from our tests. To track both objects, reporting the points around the object boundary is the most expensive. Reporting linked vertices required less resource than the point boundary reports did. Similar communication costs can be observed in related approaches which report geometric representation of object boundary [6]. As we expected, the difference between the two types of communication costs was approximately scaled by D_{split}. The ratios of reporting linked vertices against boundary points to track the two objects were both around 0.6. Combined the communication to maintain the deformable curves, the total communication cost of tracking deformable curves was still less than the cost of reporting boundary points. We also collected the communication messages to just report the vertex changes (i.e., the movement, addition and removal of vertex). To report the incremental change of object boundary,
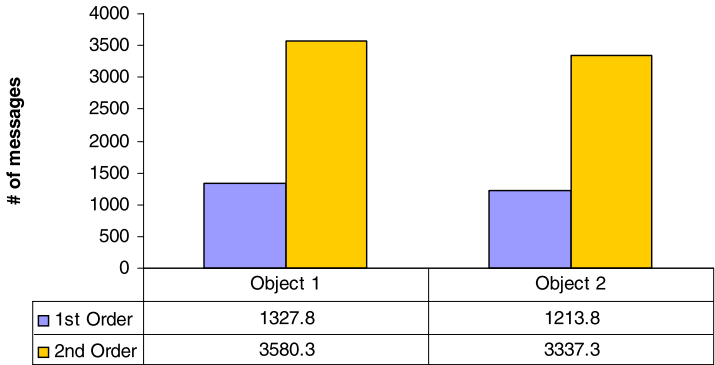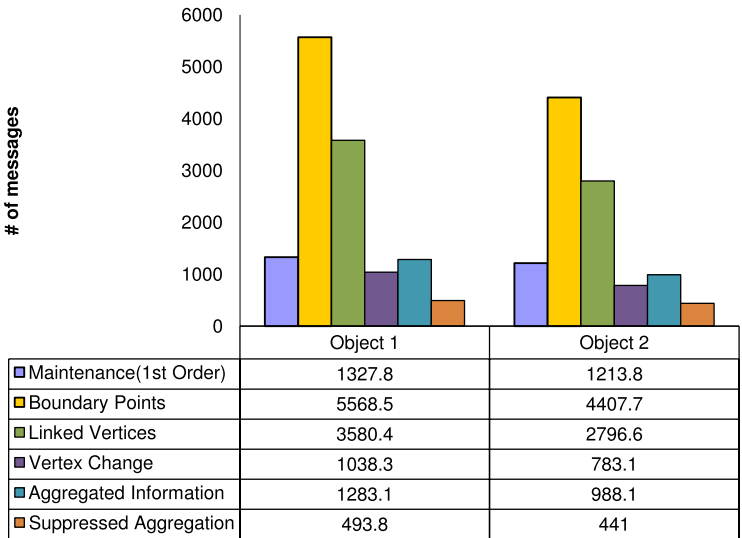
Fig. 12 Maintenance cost

	Object 1	Object 2
1st Order	1327.8	1213.8
2nd Order	3580.3	3337.3

Fig. 13 Communication cost

	Object 1	Object 2
Maintenance(1st Order)	1327.8	1213.8
Boundary Points	5568.5	4407.7
Linked Vertices	3580.4	2796.6
Vertex Change	1038.3	783.1
Aggregated Information	1283.1	988.1
Suppressed Aggregation	493.8	441

over 70% communication can be saved. Even combined with the maintenance cost, the total communication cost of reporting the incremental boundary change was less than the cost of reporting the whole geometric boundary periodically. Tracking deformable curves supports extracting abstract spatiotemporal properties about objects. We implemented the aggregation operations to extract the area and centroid of 2D objects. As shown in Fig. 13, processing the aggregated information consumed much less communication resources than reporting boundary points or linked vertices did, while these aggregated abstract spatiotemporal properties are able to assist people to understand the underlying phenomena without using any geometric representation. We also implemented the area change and centroid change operations to test lossless suppression and further improve the aggregation efficiency. In short, if the local par-
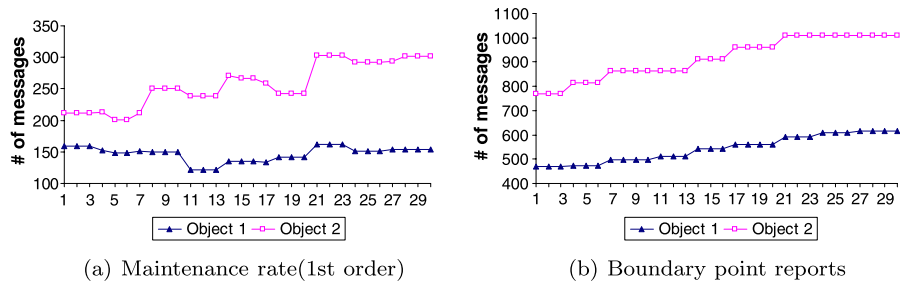
(a) Maintenance rate(1st order)   (b) Boundary point reports

Fig. 14 Communication rates

tial aggregated area change or centroid change is zero, a sensor node suppresses the local partial result to be further transmitted to its parent node. As shown in Fig. 13, the suppressed aggregation operations consumed as around 40% communication cost as required by the unsuppressed aggregation operations. This result proves the effectiveness of suppression, and shows the future direction to combine other suppression technologies with our approaches [33].

Another interesting test is the comparison of communication rates over time. Figure 14 shows the results from two tests. The object 1 moved and kept the area constant while the object 2 increased its area and kept the center still. As illustrated by Fig. 14(a), to track the object 1, the maintenance cost rate remained constant. The communication cost rate to maintain the deformable curve for the object 2 increased since more vertices were added to track the enlarged region, as shown by Fig. 14(a). Figure 14(b) explains that more communication resource was required to report boundary points of both objects while time elapsed. The object 1 required the same number of points to represent the boundary. While the object 1 moved further away from the root node's location, the boundary points required more hops to be relayed back to the root node. The object 2 needed more points to represent the boundary while the area was enlarged. More communication resources were in need to report the increasing number of points along the boundary of object 2. Other types of communication messages for linked vertices and the aggregated information showed similar results as presented in Fig. 14(b).

In the second set of tests, we used additional video clips as the underlying phenomenon to test the interactions among multiple objects. These video clips contained multiple 2D objects. The shape of these objects are complex. Convex objects, concave objects and objects with hole were all simulated. Each frame was a snapshot of these objects. A video frame was updated to TOSSIM in every 600 seconds. Sensor nodes were woken up in every 200 seconds to collect updated sensor readings, detect objects and boundaries, deform the tracking curves, and adapt curves to the topological changes. D_{split} was set to 15 for these tests. The experimental results have shown that our tracking algorithm is able to successfully track these complex objects, and adapt the curves to track multiple objects.

The first video contained a single 2D object located at the network center initially. Afterwards, the 2D object split into two objects. The two 2D objects started moving towards two opposite corners of the 100×100 space. Later on, the two 2D objects
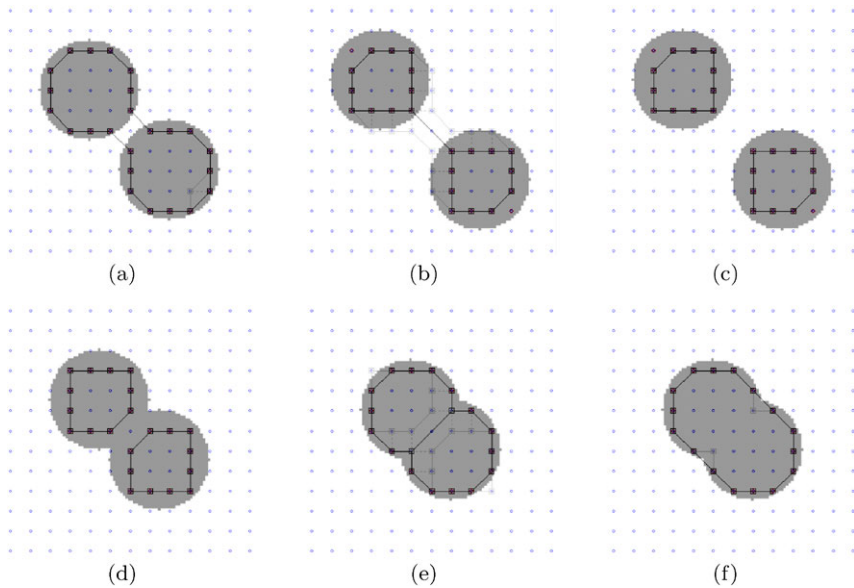
Fig. 15 Test results on splitting and merging

moved back to the center and merged into a single object. Figure 15 illustrates a series of snapshots of the tracking curves and underlying 2D objects. In Fig. 15, the gray region indicates the region covered by underlying 2D objects. The small blue circles represent the location of sensor nodes. The red circles indicate the sensor nodes which detected the object boundary. The small squares represent the vertices on deformable tracking curves. The black lines indicate the edges of the deformable curves at the current time slot, while the gray lines are the edges of the deformable curves at the previous time slot. The dotted gray lines represent the vertex movement. Figures 15(a), 15(b) and 15(c) show the sequence of the splitting event. As shown in Fig. 15(b), the sensor nodes can detect the overlapping edges locally. By removing the overlapping edges and reconnecting the open curves, sensor nodes can locally adapt the deformable curves into two closed curves as illustrated by Fig. 15(c). The sequence of the merging event is explained by Figs. 15(d), 15(e) and 15(f). Similar to Fig. 15(b), when the two 2D objects were merging together, some edges in the two closed curves overlapped together as shown by Fig. 15(e). The removal of overlapping edges can allow sensor nodes to locally adapt the deformable curves into a single closed curve, as explained by Fig. 15(f).

We used the second video to illustrate the development of a hole. The video used for the second test began with a single 2D object located at a corner in the network. Afterwards, the 2D object grew two arms both horizontally and vertically. The two arms merged at the opposite corner, which resulted in a hole inside the 2D object. Figures 16(a), 16(b) and 16(c) show the sequence of how the two arms merged. Similar to Fig. 15(e), the edges of two arms overlapped partially as illustrated by 16(b). By removing the overlapping edges and reconnecting open curves, sensor nodes can adapt the deformable curves locally to represent the ring shape of the 2D object as
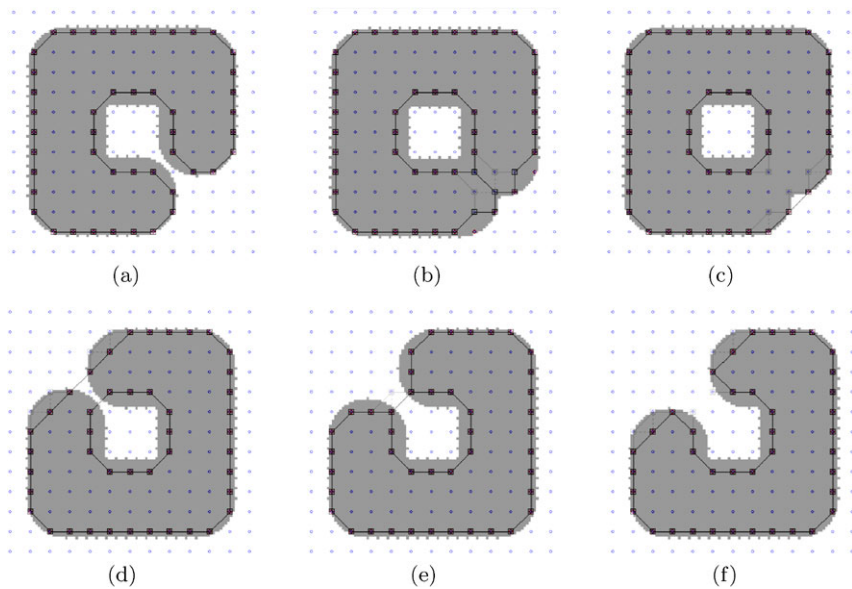
Fig. 16 Test results on a hole development

shown by Fig. 16(c). After the establishment of the inner hole, the ring started breaking at one corner. The breaking sequence is illustrated by Figs. 16(d), 16(e) and 16(f). Again, sensor nodes can locally detect the overlapping edges as shown by Fig. 16(e). Through removing the overlapping edges, the deformable curve can adapt its shape locally to the shape of underlying 2D object as illustrated by Fig. 16(f).

8 Related work

The database community treats a WSN as a distributed sensor database system which observes the physical world in real time and answers queries about the world simultaneously. Studies in sensor databases started to support complex queries with aggregation queries. TAG is a framework for aggregation query processing [26]. In TAG, sensor nodes communicate with each other in a tree structure. A sensor node prepares its local readings, and aggregates the local readings with the partial results from its children into a new local partial result. A partial result is usually kept in a constant sized message. The aggregated information is processed from the bottom leaf nodes to the top root node. The root node is able to provide the final aggregation results about the whole network.

A WSN can identify objects based on user defined thresholds [3, 6, 15, 20, 30]. To estimate the local object boundary, sensor readings are usually exchanged and processed in a neighboring region based on a statistical model. The point object boundary reports can be linked into curves in WSN. The boundary gradient information can be used to link the points. Suppose the Voronoi diagram based on sensor node location is available. By comparing the neighboring nodes' local object and

object boundary estimation results, a sensor node can find the direction of how the object boundary goes across the local Voronoi cell [25]. In this way, a local boundary point can be extended to a directed line. By connecting the partial boundary lines, a WSN can find the closed curves to represent 2D objects. Similar work has also been applied in the grid layout [34]. The skeleton structure can also be used to link the boundary points. To find a skeleton to represent the object boundary, a WSN needs to find the band region covered by the points along object boundary [37]. Based on the boundary band region, the skeleton of the region can be found to represent the object boundary.

In the field of computer vision, the deformable curve model, also known as the SNAKE model, is generally used to identify objects and object boundaries [1, 4, 18, 35, 36]. For example, the deformable curves can identify reconstructed roads from remote sensing images [1]. Recent studies in WSN have changed the focus onto how to identify complex objects, especially through the object's spatial properties. The original SNAKE model was also extended to support identifying and tracking multiple 2D objects in image and video [21, 27]. Most models require a centralized analysis of vertices and edges on the deformable curves. For example, in [21], vertices that are not located on the object boundary need to be removed. The remained open curves are reconnected through a centralized analysis. Based on a simple rectangle boundary representation, the emerging of an object can be identified through the shape matching [36]. For example, the emerging of a gas leak can be identified by the pyramid shape. Combining the tracking of deformable curves with identifying 2D objects gives us the motivation to extract complex spatiotemporal properties of 2D objects in the network. Except for the identification of a 2D object, we want to know where, when and how the 2D object evolves in spatiotemporal spaces. In [16], we have presented the initial work on applying the SNAKE model in WSN.

9 Conclusion and future work

We developed a distributed algorithm of tracking 2D objects in WSN. Based on the revised SNAKE model, our tracking algorithm maintains the deformable curve by exchanging messages around nearby sensor nodes. Furthermore, our tracking algorithm can track multiple objects and complex objects. Our simulation results showed that the maintenance cost of our tracking algorithm is relaxed and linearly scaled by the number of vertices on the curve. Based on the in-network tracking of deformable curves, many useful abstract information can be generated based on the in-network aggregation. We also presented several aggregation variations to suppress local partial aggregation results and further reduce the communication consumption. Our experiments verified the effectiveness of combining these types of aggregation operations with suppression techniques.

We will implement our tracking algorithm in real WSNs, and use more complex phenomena to test it. Our future research also needs to provide a more robust implementation of the proposed tracking algorithm for the failure-prone WSNs.

References

1. Agouris, P., Stefanidis, A., Gyftakis, S.: Differential snakes for change detection in road segments. Photogramm. Eng. Remote Sens. **67**, 1391–1399 (2001)
2. Antoine-Santoni, T., Santucci, J.F., de Gentili, E., Silvani, X., Morandini, F.: Performance of a protected wireless sensor network in a fire, analysis of fire spread and data transmission. Sensors **9**(8), 5878–5893 (2009). doi:10.3390/s90805878. URL http://www.mdpi.com/1424-8220/9/8/5878
3. Chintalapudi, K., Govindan, R.: Localized edge detection in sensor fields. Ad Hoc Netw. **1**(2–3), 273–291 (2003)
4. Cohen, L.D.: On active contour models and balloons. CVGIP: Image Understanding **53**(2), 211–218 (1991). doi:10.1016/1049-9660(91)90028-N
5. Demirbas, M., Ferhatosmanoglu, H.: Peer-to-peer spatial queries in sensor networks. In: P2P '03: Proceedings of the 3rd International Conference on Peer-to-Peer Computing, p. 32. IEEE Computer Society, Washington (2003)
6. Ding, M., Chen, D., Xing, K., Cheng, X.: Localized fault-tolerant event boundary detection in sensor networks. Proc. IEEE **2**(2), 902–913 (2005). INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies
7. Duckham, M., Nittel, S., Worboys, M.: Monitoring dynamic spatial fields using responsive geosensor networks. In: ACM GIS '05: Proceedings of the 13th Annual ACM International Workshop on Geographic Information Systems, pp. 51–60. ACM Press, New York (2005). doi:10.1145/1097064.1097073
8. Elson, J.E.: Time synchronization in wireless sensor networks. Ph.D. thesis, University of California Los Angeles, Los Angeles, CA (2003)
9. Farah, C., Zhong, C., Worboys, M., Nittel, S.: Detecting topological change using wireless sensor networks. In: Fifth International Conference on Geographic Information (GIScience 2008) (2008)
10. Galton, A.: Qualitative Spatial Change. Oxford University Press, Oxford (2000)
11. Gandhi, S., Hershberger, J., Suri, S.: Approximate isocontours and spatial summaries for sensor networks. In: IPSN '07: Proceedings of the 6th International Conference on Information Processing in Sensor Networks, pp. 400–409. ACM Press, New York (2007). doi:10.1145/1236360.1236411
12. Harrington, B., Huang, Y.: In-network surface simplification for sensor fields. In: GIS '05: Proceedings of the 13th Annual ACM International Workshop on Geographic Information Systems, pp. 41–50. ACM Press, New York (2005). doi:10.1145/1097064.1097072
13. Hellerstein, J.M., Hong, W., Madden, S., Stanek, K.: Beyond average: toward sophisticated sensing with queries. In: Second International Workshop on Information Processing in Sensor Networks, IPSN 2003, pp. 63–79 (2003)
14. Jiang, J., Worboys, M.: Event-based topology for dynamic planar areal objects. Int. J. Geogr. Inf. Sci. **23**(1), 33–60 (2009). doi:10.1080/13658810802577247
15. Jin, G., Nittel, S.: One adaptive event and event boundary detection algorithm for noisy wireless sensor networks. In: Workshop on Mobile Location-Aware Sensor Networks, Nara, Japan (2006)
16. Jin, G., Nittel, S.: Tracking deformable 2d objects in wireless sensor networks. In: GIS '08: Proceedings of the 16th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, pp. 1–4. ACM Press, New York (2008). doi:10.1145/1463434.1463517
17. Karp, B., Kung, H.T.: Gpsr: greedy perimeter stateless routing for wireless networks. In: MobiCom '00: Proceedings of the 6th Annual International Conference on Mobile Computing and Networking, pp. 243–254. ACM Press, New York (2000). doi:10.1145/345910.345953
18. Kass, M., Witkin, A., Terzopoulos, D.: Snakes: active contour models. Int. J. Comput. Vision **V1**(4), 321–331 (1988). doi:10.1007/BF00133570. URL http://dx.doi.org/10.1007%2FBF00133570
19. Kim, Y.J., Govindan, R., Karp, B., Shenker, S.: Geographic routing made practical. In: NSDI'05: Proceedings of the 2nd Conference on Symposium on Networked Systems Design & Implementation, pp. 217–230. USENIX Association, Berkeley (2005)
20. Krishnamachari, B., Iyengar, S.: Distributed Bayesian algorithms for fault-tolerant event region detection in wireless sensor networks. IEEE Trans. Comput. **53**(3), 241–250 (2004)
21. Lefèvre, S., Vincent, N.: Real time multiple object tracking based on active contours. In: Campilho, A.C., Kamel, M.S. (eds.): International Conference on Image Analysis and Recognition, ICIAR 2004, vol. 3212, pp. 606–613. Springer, Berlin (2004)
22. Levis, P., Lee, N., Welsh, M., Culler, D.: Tossim: accurate and scalable simulation of entire tinyos applications. In: SenSys '03: Proceedings of the 1st International Conference on Embedded Networked Sensor Systems, pp. 126–137. ACM Press, New York (2003). doi:10.1145/958491.958506

23. Li, J., Jannotti, J., De Couto, D.S.J., Karger, D.R., Morris, R.: A scalable location service for geographic ad hoc routing. In: Proceedings of the 6th ACM International Conference on Mobile Computing and Networking (MobiCom '00), Boston, Massachusetts, pp. 120–130 (2000)

24. Lin, C.Y., Peng, W.C., Tseng, Y.C.: Efficient in-network moving object tracking in wireless sensor networks. IEEE Trans. Mob. Comput. **5**(8), 1044–1056 (2006). doi:10.1109/TMC.2006.115

25. Liu, Y., Li, M.: Iso-map: Energy-efficient contour mapping in wireless sensor networks. In: ICDCS '07: Proceedings of the 27th International Conference on Distributed Computing Systems, p. 36. IEEE Computer Society, Washington (2007). doi:10.1109/ICDCS.2007.115

26. Madden, S., Franklin, M.J., Hellerstein, J.M., Hong, W.: Tag: a tiny aggregation service for ad-hoc sensor networks. SIGOPS Oper. Syst. Rev. **36**(SI), 131–146 (2002). doi:10.1145/844128.844142

27. McInerney, T., Terzopoulos, D.: T-snakes: topology adaptive snakes. Med. Image Anal. **4**, 73–91 (2000)

28. Ni, L.M., Liu, Y., Lau, Y.C., Patil, A.P.: Landmarc: indoor location sensing using active rfid. In: PERCOM '03: Proceedings of the First IEEE International Conference on Pervasive Computing and Communications, p. 407. IEEE Computer Society, Washington (2003)

29. Nittel, S.: A survey of geosensor networks: Advances in dynamic environmental monitoring. Sensors **9**(7), 5664–5678 (2009). doi:10.3390/s90705493

30. Nowak, R., Mitra, U.: Boundary estimation in sensor networks: theory and methods. In: IPSN, pp. 80–95 (2003)

31. Perrin, D.P., Smith, C.E.: Rethinking classical internal forces for active contour models. In: IEEE Conference on Computer Vision Pattern Recognition, vol. 2, p. 615. IEEE Computer Society, Los Alamitos (2001). doi:10.1109/CVPR.2001.991020

32. Sharifzadeh, M., Shahabi, C.: Supporting spatial aggregation in sensor network databases. In: GIS '04: Proceedings of the 12th Annual ACM International Workshop on Geographic Information Systems, pp. 166–175. ACM Press, New York (2004). doi:10.1145/1032222.1032248

33. Silberstein, A., Braynard, R., Yang, J.: Constraint chaining: on energy-efficient continuous monitoring in sensor networks. In: SIGMOD Conference, pp. 157–168 (2006)

34. Solis, I., Obraczka, K.: Efficient continuous mapping in sensor networks using isolines. In: The Second Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services (MobiQuitous 2005), San Diego, CA, USA, pp. 325–332 (2005)

35. Xu, C., Prince, J.L.: Snakes, shapes, and gradient vector flow. IEEE Trans. Image Process. **7**(3), 359–369 (1998)

36. Xue, W., Luo, Q., Chen, L., Liu, Y.: Contour map matching for event detection in sensor networks. In: SIGMOD '06: Proceedings of the 2006 ACM SIGMOD international conference on Management of data, pp. 145–156. ACM Press, New York (2006). doi:10.1145/1142473.1142491

37. Zhu, X., Sarkar, R., Gao, J., Mitchell, J.S.B.: Light-weight contour tracking in wireless sensor networks. In: The 27th Annual IEEE Conference on Computer Communications (INFOCOM'08) (2008)