Component Based Localization in Sparse Wireless Ad Hoc and Sensor Networks

Xiaoping Wang, Jun Luo, Shanshan Li, Dezun Dong, Weifang Cheng
School of Computer Science, National University of Defense Technology, Changsha, Hunan, PRC
{xiaopingwang, junluo, shanshanli, dong, wfangch}@nudt.edu.cn

Abstract— **Localization is crucial for wireless ad hoc and sensor networks. As the distance-measurement ranges are often less than that of the communication range for many ranging systems, most communication-dense wireless networks are often localization-sparse. Consequently, most existing algorithms fail to provide accurate localization supports. In order to address this issue, by introducing a concept of *component*, we propose to group nodes into components, so that nodes are able to better share their ranging and anchor knowledge. This design, CALL, relaxes two essential restrictions in localization: node ordering and anchor distribution. We evaluate the effectiveness of CALL through extensive simulations. The results show that CALL locates 80% nodes in a network with average degree 7.5 and 5% percent anchors, which outperforms the state of the art design Sweeps about 20%.**

I. INTRODUCTION

Location in wireless sensor networks (WSNs) is critical for both network operation and data interpretation [1]. Practically, it is difficult to equip each sensor with a positioning device. Instead, only a few sensor nodes, called *anchors*, know their locations, and other nodes estimate their locations through inter-node measurements from anchors [2]. Most existing localization algorithms require the network have a high density and sufficient anchors before the nodes can be accurately localized. Localization in sparse networks with few anchors is not fully addressed [3]. Indeed, a sparse network for localization is often dense in communication, because the distance-measurement ranges are typically much less than that of communication range for many ranging systems [4].

Eren and Goldenberg et al. [5, 6] investigate the theoretical conditions for network localization. They show that a network can be uniquely localized if and only if its corresponding grounded graph [5] is globally rigid and with at least three anchors embedded. The sufficient and necessary conditions for a graph to be globally rigid are: the graph is redundantly rigid and triconnected. Together with three anchors embedded, those three conditions are presented as RRT-3B [6]. RRT-3B is able to effectively test the localizability of a whole network. Further, it also identifies the localizable nodes set under general graph assumption, which makes RRT-3B a criterion to evaluate the efficacy of localization algorithms. To the best of our knowledge, there are no localization algorithms that can explicitly achieve the amount of localizable nodes as RRT-3B conditions.

The state of the art approximation algorithm to RRT-3B capability is Sweeps [3]. Sweeps utilizes the concept of finite localization to relax the node participating conditions from trilateration to bilateration. Due to its dependence on each single node to estimate the localizability locally, however, Sweeps suffers the restriction on the anchor distribution. If no nodes can find at least two anchors among its direct neighborhood, the algorithm cannot be initialized. Further, Sweeps localizes nodes by bilateration to "swept" nodes, which necessarily conduct localizing nodes spreading along the sequence of bilaterations [3]. When no nodes can perform the bilateration operation, the procedure stops, and even localizable nodes still exists.

Such limitations results in more consequences in localization-sparse networks. To address this issue, we propose a Component-bAsed Localization aLgorithm, CALL. CALL groups nodes into small components, and then merges the components recursively into a bigger one until it is realizable in a plane. By offering many advantages over node-based ones, such as favoring the large-scale localization information aggregation, facilely exploiting the anchor proximity, CALL can locate almost all localizable nodes which RRT_3B identifies. It does not rely on any certain network communication or ranging models, and neither requires specified anchor distribution.

Major contributions of this work are as follows.

1) We introduce the concept of component, and present a component-based algorithm CALL, which improves the proportion of localized nodes in sparse networks.

2) We propose a mechanism called *finite mergence* to stitch components for general component-based localization method. Further, we relax the anchor requirement conditions to facilitate component localization.

3) We conduct large-scale simulations to testify the efficiency of CALL. The results show that this algorithm can accurately locate 80% nodes in a network with average degree 7.5 by 5% percent anchors, which outperforms Sweeps about 20% in average.

The rest of this paper is organized as follows. We first discuss existing studies in Section II, and then describe the design of component-based localization algorithm in Section III. We build the theoretical foundations in Section IV, and discuss some critical issues in Section V. Finally, we evaluate the performance of CALL in Sections VI, and conclude this work in Section VII.

1-4244-2507-5/08/$20.00 ©2008 IEEE

II. Related Work

Recently, there are a plethora of works on localization in WSNs [7], falling into two categories: range-based [3, 8-14] and range-free [1, 15]. Range-free algorithms do not rely on measurement techniques but normally require high network density to approximate distance by connectivity [1]. Our work focuses more on range-based designs for sparse networks, so we briefly review the classical range-based algorithms in this section.

Savarese et al. [9] proposed a virtual coordinate based algorithm TERRAIN to solve the sparse anchor problem. TERRAIN adopts virtual coordinates and takes the advantage of the property that the virtual coordinate holds the distance information between each node pair. The essential principle used by TERRAIN is triangulation. By using virtual coordinates on each anchor, TERRAIN extends the ranging distance of anchors and makes each node triangulate to the enlarged anchors. In section III, we will show that CALL needs far less information to realize or aggregate nodes than all trilateration based algorithms, including TERRAIN. Thus CALL performs much better than TERRAIN in sparse networks.

Some researchers utilized local maps to localize nodes [10, 11]. They first use distance measurements between neighboring nodes to construct local maps, and then stitch them together to form a global map. Intuitively, components in CALL share similar notions with the local map. Nevertheless, the two methods are different greatly in the manner of mergence. In local map based methods, two maps are integrated by their common nodes, which means the common nodes need to be prior localized in their respective map. Hence, it is hard to stitch local maps in sparse networks for the severe restriction of common nodes. Comparatively, in CALL, components are merged by adjacent edges between component pair, which is easy to fulfill. Such difference affects the success of mergence as well as the applicability in sparse networks.

Savvides et al. [12] attempted to reduce the information requirement. They use collaborative multilateration among neighbors to remedy the ranging information shortage, which localizes nodes by forming an over-determined system of equations with a unique solution set. Collaborative multilateration performs better than trilateration in sparse networks. The disadvantage is that the collaboration is restricted in neighbors, so that the performance gain is limited.

As a pioneering work, Goldenberg et al. proposed Sweeps [3], which holds all possible positions of each node and prunes incompatible ones when other nodes attend the procedure. Sweeps furthest relaxes the requirement of node-based localization and achieves pretty good results in sparse networks. Given proper anchor distribution, Sweeps is able to localize a whole globally rigid region [3], but may fail to localize other regions that contain few anchors. Comparing with Sweeps, CALL is able to identify most of such regions and proper localize almost all of the localizable ones according to RRT-3B. The localizable nodes of CALL are a superset of that of Sweeps.

III. Component Based Localization

In this section, we introduce the idea of component-based localization. We will describe the rationale of CALL in Section IV.

A. Preliminary

We assume that each node locates at the distinct physical location in a plane, and some anchor nodes are able to acquire their physical positions. Each node can accurately measure the distance to its neighbors utilizing some equipments. Suppose there are n nodes in the plane, labeled as $v_1, v_2, \cdots, v_n$, and the physical position of node v_i is denoted by P_i. The indices of k anchor nodes are notated as a_i, $i \in [1,k]$. Based on the ranging information, we generate a **ranging graph** $G =< V, E >$, where each vertex denotes a node in the network and each edge means that its two-endpoint nodes can directly get the distance information of each other. For each edge $(v_i, v_j) \in E$ $i, j \in [1,n]$, d_{ij} denotes the distance between v_i and v_j.

A **realization** of ranging graph G is a mapping $f: V \to R^2$ such that 1) $f(n_{a_i}) = P_{a_i}$ for $i \in [1,k]$, and 2) for each edge $(v_i, v_j) \in E$, $\| f(v_i) - f(v_j) \| = d_{ij}$. Analogously, we can define the concept of realization on the subgraph of G, and the only difference is that we do not differentiate the rotations, translations, and reflections of the mapping when operating on a subgraph. A node is **localizable** if and only if its image is unique for all realizations of G. A node is **finitely localizable** if and only if the number of all its possible images are finite for all realizations of G. If a localization algorithm can generate the unique result for a localizable node, we say the node is **localized** by the algorithm. If a localization algorithm can generate all of the possible positions for a finitely localizable node, we say the node is **finitely localized** by the algorithm.

B. BCALL Algorithm

There are two versions of CALL: the basic version (BCALL) and the general version (CALL). BCALL can terminate in polynomial time. In contrast, CALL cannot guarantee to terminate in polynomial time, but with better efficacy than BCALL. Both of them follow three major steps: *component generation, component mergence and component realization*. We first introduce the basic idea of component-based localization through BCALL, and then present CALL by comparing it with BCALL.

Before we begin more detailed discussion of BCALL, we first introduce the formal definition of component. Given a ranging graph G, a **component** is a group of vertices that have finite realization possibilities. A component is **globally rigid** if and only if there is a unique realization in a plane. An **isolated node** in G is a node that does not belong to any components.

1) Component generation

Component generation partitions the network into globally rigid components and isolated nodes. After component generation, all nodes either belong to a component or become isolated nodes, and each node can only join in one component.

We do not distinguish anchor nodes and general nodes in this procedure.

A component is formed by a triangle in G initially; other nodes can join the component by trilateration to the nodes in the component. By this means, a component can expand as large as possible while keeping globally rigidity.

Each component has a **local coordinate system** that indicates the relative position of each node in the component. As previously mentioned, a component is initially formed by a triangle in G. Accordingly, the local coordinate system of the component is generated according to the relative position of the initial vertices in the initial triangle. The other nodes then join in the component calculate and record their coordinates when they process trilaterations.

After component generation, components further make a decision about whether they are realizable. If a component is realizable, then it will enter component realization procedure. Otherwise, those non-realizable components will try to merge with other components and perform the component mergence procedure. Merging components potentially make some non-realizable components capable to be realized, because 1) the mergence causes the merged components to aggregate both their nodes and their anchor information, and 2) the realizing requirements for a component are independent with the number of nodes in the components.

2) Component mergence

Component mergence integrates two components into a bigger one. Only those components that fulfill the merging conditions can process component mergence. For BCALL, it requires the resultant component to be globally rigid. After merged, the local coordinate systems of the two components must be consistent. We accomplish this by converting the local coordinate system of a component to that of the other one. The merging conditions guarantee the feasibility of the conversion.

Component mergence is a recursive process, because new merging opportunities may emerge after a round of mergence. In other words, some mergence can make other components or isolated nodes capable to merge into the resultant component. Component mergence process stops when no such mergence can process or the resultant component is realizable.

3) Component realization

Component realization converts the local coordinate system of the component into the physical one. Only those components that fulfill the realizing conditions can process component realization. For BCALL, it requires the realization to be unique. The realization is done by mapping the local coordinates to physical positions.

Components are merged and realized as a whole. The nodes belonging to the same component are localized simultaneously. This is the main difference between component-based algorithms and node-based ones.

4) BCALL complexity analysis

In a macro view of BCALL, we show the integrated procedure in Algorithm 1. In Section IV, we will describe the rationale of the rules, by which BCALL integrates and realizes components.

Algorithm 1 BCALL

Input: the ranging graph G
Output: the realized nodes of G

1: Invoke component generation process to partition G into components and isolated nodes
   C = GenerateComponents(G)
   $C_1 = \varnothing$
2: **while** C is not empty and $C \neq C_1$ **do**
3: $C_1 = C$
4: Perform component realization to realize the components in C_1
   C_1 = BasicRealizeComponents(C_1)
5: Further merge components in C_1
   C = BasicMergeComponent(C_1)
6: Return the localized node set $N_L = V(G) \backslash \mathrm{V}(C)$, and their realization

GenerateComponents(G)

1: Assign component set C to be empty
2: **while** G contains triangles **do**
3: Select an arbitrary vertex included by a triangle $v \in \mathrm{V}(G)$ as sponsor node, and construct a local coordinate system from the triangle; Create the component C_v
4: **while** existing a node u in $G \backslash C_v$ neighboring to C_v, and existing trilateration between u and component C_v **do**
5: $C_v = C_v \cup u$
6: $G = G \backslash C_v$
7: $C = C \cup C_v$
8: The remainder is isolated nodes
   $C = C \cup G$
   Return C

BasicRealizeComponents(C)

1: Label anchors as realized nodes
2: Assign set N_R to be all realized nodes
3: **for** each isolate node $n \in C$ **do**
4: **if** n can find three edges link to realized nodes in N_R **then** $N_R = N_R \cup n$, $C = C \backslash n$
5: **for** each component c in C **do**
6: **if** Component c (1) contains three anchors, or (2) contains two anchors and a non-anchor node sharing an edge with a realized node; or (3) contains one anchor and two distinct non-anchor nodes sharing two edges with two distinct nodes which are realized; or (4) there are at least four independent edges connecting the component c with realized nodes. **then**
7: Label c as realized component, and $C = C \backslash c$
8: Return C

BasicMergeComponents(C)

1: **If** an isolated node n can process trilateration to a component c **then**
2: $c_1 = c \cup n$
3: $C = C \backslash c$
4: $C = C \cup c_1$
5: **If** two components c_1 and c_2 fulfills that (1) there are at least four independent edges connecting the two components; and (2) there are at least three vertices in each component associated with these edges. **then**
6: $c = c_1 \cup c_2$
7: $C = C \backslash c_1$, $C = C \backslash c_2$
8: $C = C \cup c$
9: Return C

BCALL produces globally rigid components and processes unique realization. It operates on the set C whose size is bounded by the total number of nodes n. Moreover, all steps in Algorithm 1 will strictly diminish the size of set, and can terminate in polynomial time. Hence, BCALL can terminate in polynomial time.

C. CALL Algorithm

In this section, we first introduce CALL design by comparing with BCALL, and then demonstrate CALL execution in a randomly generated network.

1) CALL design

Comparing with BCALL, CALL relaxes the constraints in two aspects. First, CALL does not demand the resultant of mergence to be globally rigid. Second, CALL does not require component realization to be unique. That is, two components can merge if the resultant component has finite realization possibilities, and a component can be realized when the number of possible realizations is finite. The aim of CALL is to realize components with best efforts, then prune the redundant possibilities when enough information available.

The relaxations cause nodes to have several possible positions. CALL records these positions in the **potential position set** on each node, which indicates the all possible coordinates of the node in the local coordinate system or physical coordinate system.

CALL adds a sub-step in each procedure to handle the potential position sets. After each mergence or realization, all nodes will prune the incompatible items in the potential position sets. **Incompatible items** are those that produce no logical results. The procedure goes on until no incompatible items exist. When neighboring components are both realized, the inconsistent items in their potential position sets will also trigger the conflict resolution procedure as mentioned.

Finally, all nodes belonging to realized components are finitely localized. The residual items in potential position set of each realized node indicates the possible positions of the node. We show the procedure of CALL in Algorithm 2. In Section IV, we will explain the reason why the rules in Algorithm 2 guarantee the mergence and realization to be finite.

Unfortunately, this procedure cannot guarantee to terminate in polynomial time, because the potential position set may grow exponentially in the worst case. We will discuss this problem in Section V.

2) Example of CALL execution

We demonstrate the CALL execution by highlighting the component mergence procedure. We take Figure 1 as an example. In the graph 100 nodes are randomly deployed in a square region under UDG model [16]. The average degree is about 6.Figure 1 shows the state after component generation. The lowercase letters represent the generated components, and the isolated nodes are labeled with numbers. We use $\{a,b\}$ represents the resultant component composed by component a and b. The bold letters denote globally rigid parts whose possible locations in the merged components are unique. We start from a randomly selected component, supposing the upper left component d. We follow the mergence procedure serially and compact several mergence operations into one step, because the merging sequence does not influence the result. The execution of component mergence is as follows.

1. Component d can merge component e and node 4 finitely, and we get $\{\boldsymbol{d},e,4\}$.

Algorithm 2 CALL

Input: the ranging graph G
Output: the finitely realized nodes of G

Invoke component generation process as BCALL to partition G into components and isolated nodes
  C = GenerateComponents(G)
  $C_1 = \varnothing$
while C is not empty and $C \neq C_1$ **do**
  $C_1 = C$
  Perform component realization to realize the components in C_1
  C_1 = RealizeComponents(C_1)
  Further merge components in C_1
  C = MergeComponent(C_1)
Return the localized node set $N_L = V(G) \backslash V(C)$, and their potential position sets

RealizeComponents(C)

Label anchors as realized nodes
Assign set N_R to be all realized nodes
for each isolate node $n \in C$ **do**
  if n can find two edges link to realized nodes in N_R **then**
    $N_R = N_R \cup n$, $C = C \backslash n$
    Update the potential position sets and prune all the incompatible items
for each component c in C **do**
  if Component c (1)contains two anchors, or (2) Contains an anchor and a non-anchor node sharing an edge with a realized node; or (3) there are at least three edges connecting c with at least two distinct nodes which are realized, and there are at least two vertices associated with these edges in c. **then**
    Label c as realized component
    Update the potential position sets and prune all the incompatible items
    $C = C \backslash c$
Return C

MergeComponents(C)

If an isolated node n can process bilateration to a component c **then**
  $C = C \backslash c$
  $c = c \cup n$, $C = C \cup c$
  Update the potential position sets and prune all the incompatible items
If two components c_1 and c_2 fulfills that (1) there are at least there independent edges connecting the two components; and (2) there are at least two vertices in each component associated with these edges. **then**
  $c = c_1 \cup c_2$
  $C = C \backslash c_1$, $C = C \backslash c_2$
  $C = C \cup c$
  Update the potential position sets and prune all the incompatible items
Return C

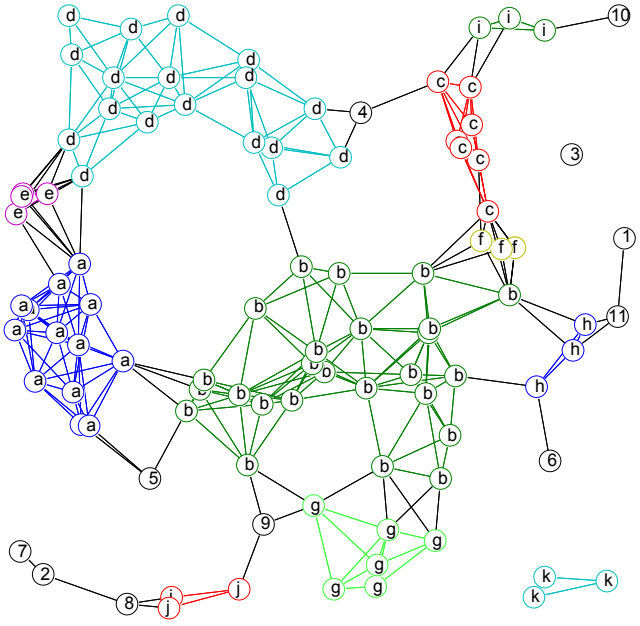


Fig. 1. Example of CALL execution.

2. The resultant component can merge component *a* finitely, thus we get {***d***,*e*,*a*,4}
3. The resultant component can merge node 5 finitely. We get {***d***,*e*,*a*,4,5}.
4. Component *b* has five edges shared with {***d***,*e*,*a*,4,5}, and it provides additional information to prune redundant items. As a result, it makes component *a*,*e* and node 5 finalized, and generates {***d***,***e***,***a***,***b***,**5**,4}.
5. The resultant component can merge component *g* uniquely and component *f*,*h* finitely. We get {***d***,***e***,***a***,***b***,***g***,*f*,*h*,**5**,4}.
6. The resultant can merge nodes 9,11 finitely and component *c* uniquely. By the information provided by component *c*, node *4* and component *f* are finalized. We get {***d***,***e***,***a***,***b***,***g***,***f***,***c***,*h*,**5**,**4**,9,11}
7. The resultant component can merge component *i* finitely. The final result is {***d***,***e***,***a***,***b***,***g***,***f***,***c***,*h*,*i*,**5**,**4**,9,11} and the component mergence procedure finishes.

After this procedure, a component {***d***,***e***,***a***,***b***,***g***,***f***,***c***,*h*,*i*,**5**,**4**,9,11} is formed. It covers most nodes in the network and judges realizable as a whole. For example, if node 4,5,1 are anchors, the final result will be {***d***,***e***,***a***,***b***,***g***,***f***,***c***,***h***,*i*,**5**,**4**,9,**11**,**1**}, in which the bold letters presents the uniquely localized nodes and others presents finitely localized nodes. If node 8 is also an anchor, then component *j* will be realized finitely. Moreover, the reflected position of node 9 cannot generate any results in this realization process, and this realization makes node 9 finalized.

IV. Theoretical Foundations

In this section, we introduce the rationale behind the rules for component mergence and component realization used in BCALL and CALL.

Lemma 1. Two globally rigid components can merge into a component if 1) there are at least three edges connecting the two components and 2) there are at least two vertices in each component associated with these edges.

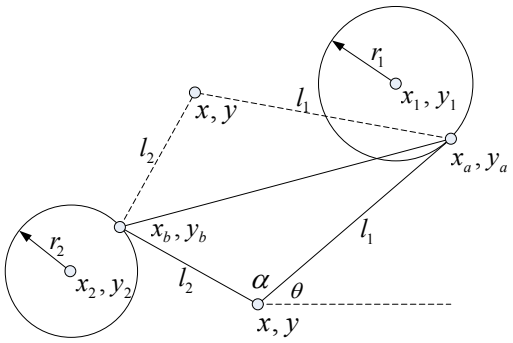


Fig. 2. Proof of component mergence.

Proof: Given two globally rigid components *A* and *B*, they are connected by three edges. We label the vertices associated with these edges 1, 2, 3 in *A* and *a, b, c* in *B*. Without loss of generality, the three edges connecting the two components are denoted as (1,*a*), (2,*b*), and (3,*c*) respectively. The distance of each edge is denoted as $r_1 = d_{1a}$, $r_2 = d_{2b}$, and $r_3 = d_{3c}$.

We merge these two components by converting the local coordinate system of *B* to that of *A*. Instead, we accomplish the conversion by converting the coordinate of vertex *a*, *b*, and *c*, and convert other vertices based on conversion of these vertices.

If node 1, 2, 3 or node *a*, *b*, *c* are not distinct nodes, then the convention can be done by a series of bilaterations. Apparently, the results are finite.

We now consider the case that node 1, 2, 3 and node *a*, *b*, *c* are distinct nodes . Let (x_1, y_1), (x_2, y_2) and (x_3, y_3) denote the original coordinates of vertex 1, 2, and 3 in the local coordinate system of *A*. Let (x_a, y_a), (x_b, y_b), and (x, y) denote the converted coordinates of vertex *a*, *b*, and *c* in the local coordinate system of *A*. The distances of *ab*, *ac*, and *bc* hold through the mergence, hence we use two distances $l_1 = d_{ac}$, $l_2 = d_{bc}$ and their angular separation α to represent this relationship. We introduce a parameter θ to simplify the expressions, which denotes the angular separation of line *ac* and the *x*-coordinate.

First, as shown in Figure 2, we ignore the edge (3,*c*). Then triangle *abc* can flip against axis *ab*, denoted by the dashed line. We take *x*, *y*, and θ as unknown variables, then the coordinates of vertex *a* and *b* can be expressed as:

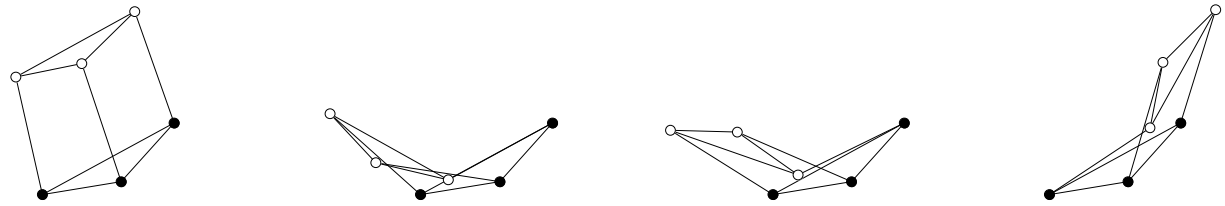
$$\begin{cases} x_a = x + l_1 \cos\theta \\ y_a = y + l_1 \sin\theta \end{cases} \tag{1}$$

$$\begin{cases} x_b = x + l_2 \cos(\theta \pm \alpha) \\ y_b = y + l_2 \sin(\theta \pm \alpha) \end{cases} \tag{2}$$

Considering the edges (1,*a*) and (2,*b*), we have:

$$\begin{cases} (x + l_1 \cos\theta - x_1)^2 + (y + l_1 \sin\theta - y_1)^2 = r_1^2 \\ (x + l_2 \cos(\theta \pm \alpha) - x_2)^2 + (y + l_2 \sin(\theta \pm \alpha) - y_2)^2 = r_2^2 \end{cases} \tag{3}$$

After expending equations in (3), we obtain binary linear simultaneous equations about $\sin\theta$ and $\cos\theta$. We use substitution as follows:



(a) (b) (c) (d)

Fig. 3. All possibilities of certain component mergence

$$\begin{cases} a = 2(x - x_1)l_1 \\ b = 2(y - y_1)l_1 \\ c = 2l_2((x - x_2)\cos\alpha \pm (y - y_2)\sin\alpha) \\ d = 2l_2((y - y_2)\cos\alpha \mp (x - x_2)\sin\alpha) \\ e = r_1^2 - l_1^2 - (x - x_1)^2 - (y - y_1)^2 \\ f = r_2^2 - l_2^2 - (x - x_2)^2 - (y - y_2)^2 \end{cases} \quad (4)$$

Eliminating the parameter θ, and we have:

$$(bf - de)^2 + (ce - af)^2 = (bc - ad)^2 \quad (5)$$

Substitute the parameters of (5) by equations in (4). We obtain an expression of x and y.

Considering the edge (3,c), we have:

$$(x - x_3)^2 + (y - y_3)^2 = r_3^2 \quad (6)$$

Solving the simultaneous equations formed by (5) and (6), we can calculate the values of x and y. Thus we can get the values of θ by equation (3). The new coordinates of a and b can also be calculated by equation (1) and (2).

It is easy to verify that equation (5) is a sextic equation and equation (6) is a quadratic equation. By Bézout's theorem [17], the maximum number of real roots of simultaneous equations (5) and (6) is twelve. Considering the flip ambiguity, the maximum number of possibilities is twenty-four. The resultant component has finite realization possibilities in a plane. ■

Actually the bound of solution number is not tight, and it can be improved by exploiting the speciality of the equation (5) and (6). Practically, the number of real roots is quite smaller than the theoretical upper bound, typically less than eight. Figure 3 shows all the possibilities of a typical component mergence.

Theorem 1 (Mergence for BCALL). If two globally rigid components can merge into one globally rigid component, then 1) there are at least four edges connecting the two components and 2) there are at least three vertices in each component associated with these edges.

Proof: By Lemma 1 at least four edges are required for generating a globally rigid component, or the resultant component may have several realization possibilities. If only two distinct vertices connect any side of the merged component, it may be ambiguous due to flipping against the axis of these two vertices. Hence, it requires at least three distinct vertices in each component to form a globally rigid component. ■

Theorems 1 provides the necessary conditions for how components can be merged and keep globally rigidity. Let us consider the example shown in Figure 4. The two components follow the conditions in Theorem 1, but the resultant component can flip against axis 1c. We say four edges *independent* if they hold globally rigidity in the mergence. In BCALL, we require the edges used for component mergence to be independent.

Theorem 2 (Finite mergence for CALL). Two components can merge into a component if 1) there are at least three edges connecting the two components and 2) there are at least two vertices in each component associated with these edges.

Proof: All vertices in each component have finite potential position possibilities. By Lemma 1, the resultant component has realization possibilities bounded by the product of each possibility i.e. the possibility of mergence and possibilities of these vertices. ■

Theorem 3 (Finite mergence for nodes). A node can be merged to a component by two edges connecting to the component.

Proof: The maximum possibilities of localizing a node by two edges are two. Hence, the potential coordinates of the node in the local coordinate system are the enumeration of all the possibilities. .■

Theorem 4 (Realization for CALL). A component can be realized to finite possibilities by fulfilling at least one of following conditions:

a) Contains two anchors;

b) Contains one anchor and a non-anchor node sharing an edge with a realized node;

c) There are at least three edges connecting the component with at least two distinct nodes which are realized, and there are at least two vertices associated with these edges in the component.

Proof: a) Each node in the component has finite distance possibilities to these two anchor nodes. By Theorem 3 they can calculate finite potential positions in physical coordinate system. Thus the component can be realized in finite possibilities.

b) The node sharing a common edge with a realized node has finite possibilities to be realized by Theorem 3. Thus this case can be converted to case a).

c) Take the physical plane as a virtual component. By Theorem 2 we can merge the component to the virtual component by three edges. The local coordinate system formed by anchors is a physical coordinate system, thus the merged components can be realized to finite possibilities. ■

Corollary 1 (Realization for BCALL). A globally rigid component can be uniquely realized by containing three distinct vertices fulfilling one of following conditions:

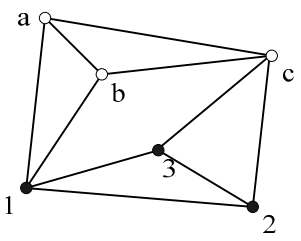


Fig. 4. Counterexample of globally rigid merging condition

a) They are three anchors;

b) They are two anchors and a non-anchor node sharing an edge with a realized node;

c) They are one anchor and two distinct non-anchor nodes sharing two edges with two distinct realized nodes;

d) There are at least four independent edges connecting them with at least three distinct realized nodes.

Proof: Each condition is generated by adding a new restriction to each condition of Theorem 4 to eliminate the ambiguity. Thus it can be concluded by Theorem 4 and Theorem 1 in each case. ■

V. Discussion

In this section, we discuss some practical issues of CALL and the solutions to address them.

A. Distributed Implementation

In practical implementations, nodes are difficult to get the global view of components in the whole network. They only know their component ID and local coordinates in the component. Nevertheless, CALL requires cooperating in two aspects behaviors: local coordinate system creation in component generation and information gathering in component mergence or component realization.

Local coordinate system creation requires compatible coordinate assignment. CALL designates the sponsor node of a component create local coordinate system according to the distances between the initial nodes. Then other nodes can join in the component by trilateration. The course can be carried out in a fully distributed fashion.

Information gathering requires components to aggregate all information about neighboring components and realized nodes. CALL adopts GHT [18] to tackle this problem. CALL hashes information by the source target id of the information, and the realized nodes shares the same id. Hence, those specified nodes can gather enough information about whether this component can be realized or merged with others. If realization or mergence is triggered, then the convention of local coordinate systems can be done by broadcasting a message that indicates the mapping rules.

B. Impact of Ranging Errors

In actual network, ranging errors always exist. CALL is not applicable in this situation, because CALL may eliminate correct item by some ambiguous values. Fortunately, we can extend BCALL to handle this problem, because rules of BCALL provide redundant information in each step to mitigate the impact of errors. Hence, we can merge and realize components by a improved version of Coordinate System Registration [11].

Another problem induced by ranging errors is that the result will decay rapidly by errors accumulation. This is in nature unsolvable by localization algorithms, thus the effective way to solve this problem is improving ranging accuracy or increasing network and anchor density.

C. Computation Complexity

As previously mentioned, CALL cannot guarantee to terminate in polynomial time, because the size of potential position set may increase exponential in the worst case. We can mitigate the explosion of set size by endowing a higher priority of such mergence that produces smaller result set. This mechanism, however, does not change the inherent complexity of CALL, and it is impractical for those resource-constrained sensor nodes to maintain huge data. Hence, it is necessary for nodes to judge before each mergence and terminate the intolerable ones, even though this method is negative for localization itself.

Computation cost of coordinate conversion is also not trivial. Lemma 1 provides a way to calculate the converted coordinates, but it may need to solve nonlinear simultaneous equations. There are no analytic solutions for the equations. However, numerical approaches using the iterative or homotopy method [19] can solve this problem efficiently. These methods does not require much resource, thus can be applied to sensor nodes.

VI. Simulation Results

A. Experiment Setup

We generate uniformly random networks of 200 nodes in a square region with diversified average neighbor number and anchor number. We use $1/\sqrt{2}$-QUDG model to get more realistic topology. Moreover, the connectivity information for distances between $1/\sqrt{2}$ and 1 is estimated by linear probability. The network connectivity is controlled by the ranging radius of each node. In order to mitigate the randomness, we repeat each experiment 20 times and report the average of outputs.

We evaluate our algorithm by comparing with two typical approaches Sweeps [3] and RRT-3B [6], which we have discussed in Section I and II. Sweeps is the most effective localization algorithm in sparse networks and it outperforms other trilateration-based algorithms much [3]. RRT-3B can identify most of localizable nodes in a general network, thus provides a criterion of localization algorithms.

We consider two metrics for performance evaluation, proportion of localized nodes and the size of potential poison set. The proportion of localized nodes shows the efficacy of each algorithm, and the size of potential poison sets indicates the cost of each algorithm. We do not evaluate the communication cost, because localization only needs one-time-execution during network startup and we can afford an acceptable communication cost in such condition.

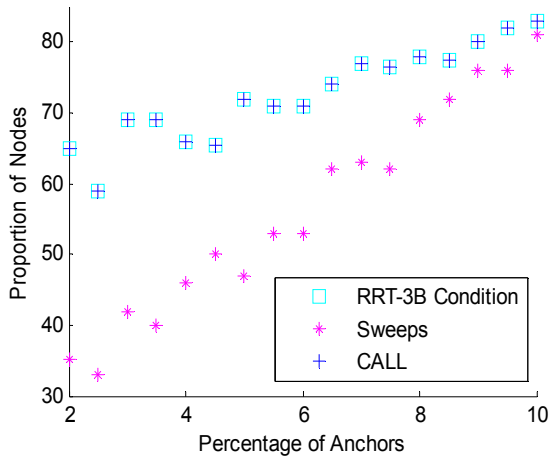


Fig. 5. Proportion of uniquely localized nodes with anchor density varying.

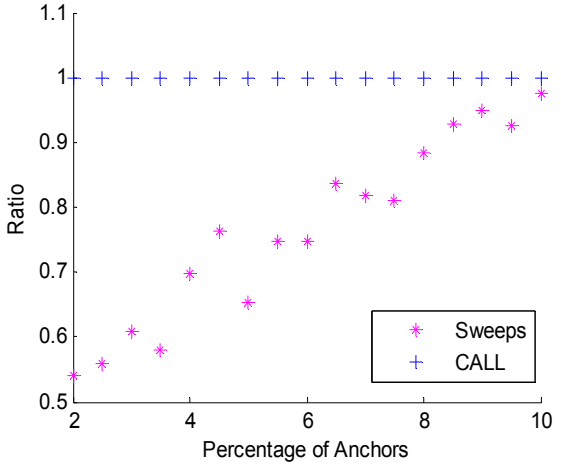


Fig. 6. Ratios to RRT-3B with anchor density varying.

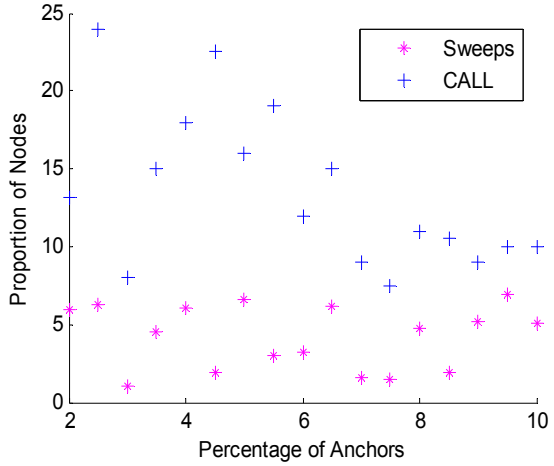


Fig. 7. Proportion of finitely localized nodes with anchor density varying.

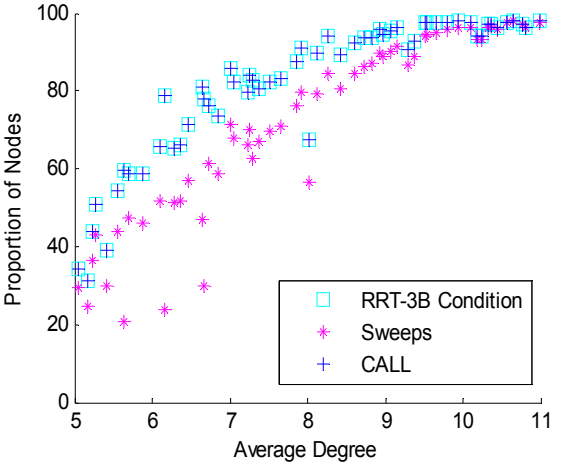


Fig. 8. Proportion of uniquely localized nodes with the average degree varying.

B. Localized Nodes Against Anchor Density

Figure 5-7 show the proportion of localized nodes by each algorithm when anchor density varies. In each case, the average degree of the generated network varies from 6 to 7. The mean value is about 6.5.

Figure 5 plots the proportion of uniquely localized nodes. As the anchor density increases, all algorithms can locate more nodes relatively. It is a bit counter-intuitive that CALL can always localize exactly the same nodes as RRT-3B identifies. This demonstrates that CALL can perform exhaustive information gathering. CALL, however, is not strictly equal to RRT-3B. An example is $K_{3,4}$, which is globally rigid but contains no trilaterations. In such graph, no component can form, but it is a localizable graph in theory. Such cases always exist when the number of vertices is above seven, and none of the sub-graphs in those graphs are globally rigid. That is to say, the only way to identify these globally rigid parts is to check all possible node sets. This procedure requires an exponential computation cost. That is why CALL does not find all globally rigid parts of a network in components generation step. Nevertheless, such examples can rarely be generated in randomly scattered networks. Hence, CALL can localize the same number of nodes as theoretical limits in most cases. CALL outperforms Sweeps especially when anchors are sparse. The difference between Call and Sweeps reduces with the increase of the percentage of anchors, and becomes slight when the percentage of anchors is over 8%.

Figure 6 is a transformed version of Figure 5. We use RRT-3B as a general metric and compare the algorithms by their localized radios to RRT-3B. Under this view, CALL is quite stable, Sweeps tends to approach CALL when anchor density enlarges. In average, the performance gain of CALL is about 20% higher than Sweeps.

Figure 7 shows the proportion of nodes with finite possibilities when the anchor density varies. CALL localizes larger number of nodes in finite states than Sweeps does, since CALL has the ability of integrating information though the whole network and can identify most finitely localizable nodes. The number of finitely localized nodes decreases when the anchor density increases. The inherent reason is that the total amount of finitely localizable nodes in a network is

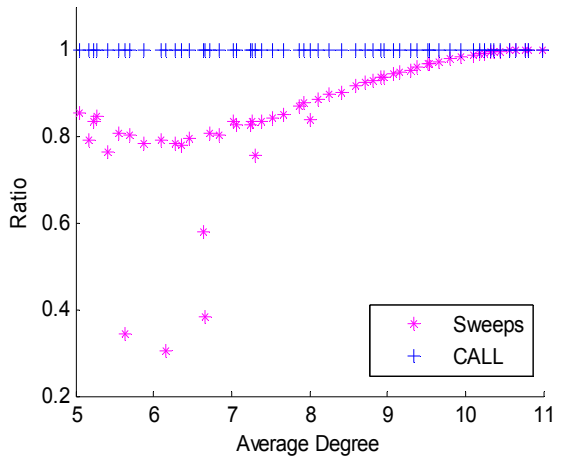


Fig. 9. Ratios to RRT-3B with the average degree varying.

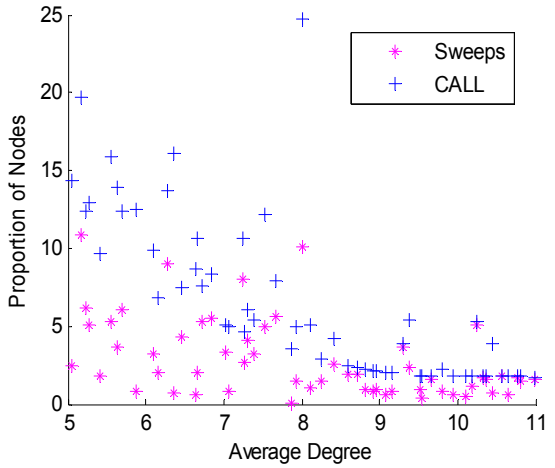


Fig. 10. Proportion of finitely localized nodes with the average degree varying.

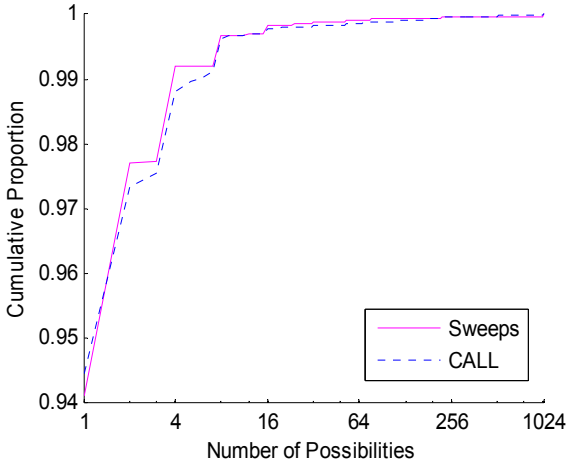


Fig. 11. Distribution of possibilities on each node.

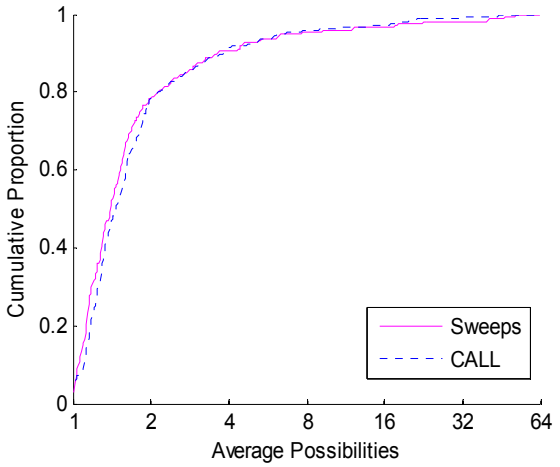


Fig. 12. Distribution of average possibilities.

reduced when more anchors exist. In contrast, the number of finitely localized nodes by Sweeps is quite noisy. This phenomenon can be observed with independent of anchor density. This can be explained as Sweeps can uniquely localize a whole globally rigid region [3] and localize nodes on the border of the region finitely. The size and number of border nodes of such regions are both random [6].

C. Localized Nodes Against Network Density

Figure 8-10 report the proportion of localized nodes by each algorithm when the average degree of the network varies. We fix the anchor density in 5%.

Figure 8 shows the proportion of uniquely localized nodes. As the average degree increases, all algorithms locate more nodes relatively. When the average degree is greater than 10, the improvement of CALL turns to be trivial. This is an experimental boundary of whether CALL is applicable. It is also approved in this figure that CALL can locate almost the same number of nodes as RRT-3B identifies.

We also plot the ratios of each algorithm to the RRT-3B in Figure 9. From this figure, we can conclude that Sweeps performs better when network density enlarges, because Sweeps specializes in localizing a single globally rigid region and can localize most of nodes when the whole network is globally rigid. The two algorithms make little difference when average degree is over 10.

Figure 10 plots the proportion of nodes with finite possibilities when the average degree varies. Herein we observe that a higher network density leads to a lower proportion of finitely localizable nodes. That is because nodes have higher probabilities to form globally rigid components in denser networks, and localizing nodes in globally rigid components cause no ambiguities at all.

D. Distribution of Possibilities

The worst-case complexity of all algorithms could be exponential in the number of nodes. We investigate this problem by our simulation results and report the results in Figure 11-13. Note that the x-coordinate in each figure is all logarithmic.

Figure 11 shows the cumulative distribution of possibilities of each single node. All the algorithms uniquely localize over 94% of nodes. CALL uniquely locates higher proportion of nodes than Sweeps does. Moreover, CALL also locates

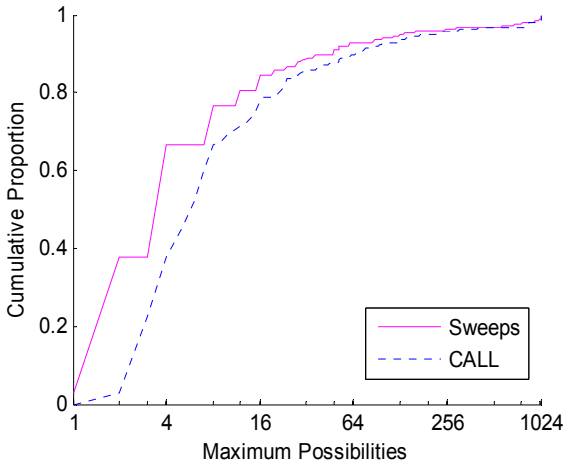


Fig. 13. Distribution of maximum possibilities.

nodes into more possible positions than Sweeps does. The reason is that CALL can locate more nodes in both finite states and unique states as shown in Figure 6 and Figure 7.

Figure 12 shows the cumulative distribution of the average location possibilities of each network instance. The average possibilities of networks in each algorithm are less than 4 in about 90% cases. For average degree, the fluctuation of CALL is milder than that of Sweeps. That is because the average possibility of CALL is averaged by the number of nodes it localized.

Figure 13 plots the cumulative distribution of the maximum location possibilities in each network instance. The maximum possibilities of networks in each algorithm are less than 64 in over 90% cases. CALL gets higher maximum possibilities because CALL finitely localizes more nodes than Sweeps does, and the combinatorial explosion becomes more serious when operating on a larger set. A combined view of the three figures suggests that the costs of CALL and Sweeps are of the same level in all aspects.

To summarize, CALL locates almost all nodes as RRT-3B, and outperforms Sweeps about 20% in average. At the same time, the cost of CALL and Sweeps is indistinctive.

VII. Conclusions

We present the concept of component and propose a component-based approach, CALL, to address the localization issue in sparse wireless ad hoc and sensor networks. We form basic rules for operations on components. Simulation results show that this design significantly outperforms previous designs. The future work leads into several directions. First, we will deal with noisy ranging issue. Second, we are going to investigate the theoretical bound of localizability using polynomial spatial-temporal cost.

Acknowledgement

This work is supported in part by the National Basic Research Program of China (973 Program) under grant No. 2006CB303000 and NSFC projects under grant No. 60621003.

Reference

[1] S. Lederer, Y. Wang, and J. Gao, "Connectivity-based Localization of Large Scale Sensor Networks with Complex Shape," in *INFOCOM*, 2008.

[2] N. Bulusu, J. Heidemann, D. Estrin, and T. Tran, "Self-configuring Localization Systems: Design and Experimental Evaluation," *ACM Transactions on Embedded Computer Systems,* 2003.

[3] D. Goldenberg, P. Bihler, M. Cao, J. Fang, B. Anderson, A. S. Morse, and Y. R. Yang, "Localization in Sparse Networks using Sweeps," in *MobiCom*, 2006.

[4] C. Peng, G. Shen, Y. Zhang, Y. Li, and K. Tan, "BeepBeep: A High Accuracy Acoustic Ranging System using COTS Mobile Devices," in *ACM Sensys*, 2007.

[5] T. Eren, D. K. Goldenberg, W. Whiteley, Y. R. Yang, A. S. Morse, B. D. O. Anderson, and P. N. Belhumeur, "Rigidity, Computation, and Randomization in Network Localization," in *INFOCOM*, 2004.

[6] D. K. Goldenberg, A. Krishnamurthy, W. C. Maness, Y. R. Yang, and A. Young, "Network Localization in Partially Localizable Networks," in *INFOCOM*, 2005.

[7] A. Srinivasan and J. Wu, "A Survey on Secure Localization in Wireless Sensor Networks," *Encyclopedia of Wireless and Mobile Communications,* 2008.

[8] Z. Yang and Y. Liu, "Quality of Trilateration: Confidence-based Iterative Localization," in *IEEE ICDCS*, 2008.

[9] C. Savarese, K. Langendoen, and J. Rabaey, "Robust positioning algorithms for distributed ad-hoc wireless sensor networks," in *USENIX Annual Technical Conference*, 2002.

[10] X. Ji and H. Zha, "Sensor Positioning in Wireless Ad-hoc Sensor Networks with Multidimensional Scaling," in *INFOCOM*, 2004.

[11] D. Moore, J. Leonard, D. Rus, and S. Teller, "Robust Distributed Network Localization with Noisy Range Measurements," in *SenSys*, 2004.

[12] A. Savvides, H. Park, and M. B. Srivastava, "The bits and flops of the n-hop multilateration primitive for node localization problems," in *International Workshop on Sensor Networks Application*, 2002, pp. 112–121.

[13] Z. Yang, M. Li, and Y. Liu, "Sea Depth Measurement with Restricted Floating Sensors," in *RTSS*, 2008.

[14] L. Xiao, L. J. Greenstein, and N. B. Mandayam, "Sensor-Assisted Localization in Cellular Systems," *Transactions on Wireless Communications,* 2007.

[15] M. Li and Y. Liu, "Rendered Path: Range-Free Localization in Anisotropic Sensor Networks with Holes," in *ACM MobiCom*, 2007.

[16] J. Bruck, J. Gao, and A. A. Jiang, "MAP: Medial Axis Based Geometric Routing in Sensor Network," in *MobiCom*, 2005.

[17] G. Fischer, *Plane algebraic curves* vol. 15: American Mathematical Society, 2001.

[18] S. Ratnasamy, B. Karp, L. Yin, F. Yu, D. Estrin, R. Govindan, and S. Shenker, "GHT: A Geographic Hash Table for DataCentric Storage," in *ACM International Workshop on Wireless Sensor Networks and Applications (WSNA)* Atlanta, 2002, p. 78~87.

[19] T. POMENTALE, "Homotopy iterative methods for polynomial equations," *IMA Journal of Applied Mathematics,* vol. 13, pp. 201–213, 1974.