Implementing a noise-tolerant localization algorithm

David Bozarth
Engineering Science Department, Sonoma State University, Rohnert Park, CA 94928

A published algorithm for localizing wireless sensor nodes in a planar network was partially implemented in a C++ design. The algorithm features tolerance to communication noise and accurate localization of any nodes that meet readily determined criteria. The implementation succeeded through the first step of a three-step process.

Introduction

In many sensor network applications, data acquired from the network must include a reliable spatial component in order to be useful. Therefore information on the relative locations of network nodes is needed. If the network is deployed in a non-deterministic manner – its components being dropped to the ground from an aircraft, for example – then localization becomes a critical aspect of initializing the network.

For low-cost wireless sensors with limited communication range, it is desirable to design localization capability into the network itself. Information required to map the network must be obtained, passed and processed among the sensor nodes. Assuming the network nodes are all the same type of device, and the device communication range is at least twice the sensing range, then sensor coverage of an area implies network connectivity over the area [1].

Typically localization of a large network is built up in tree fashion from local clusters, with processed information flowing toward the root [2]. The basic currency of this process is range information.

Problem scenario

A network of wireless sensors is arranged in two dimensions. Their relative locations are unknown. Range measurements can be made by signaling between sensors, but these signals may contain noise sufficient to introduce ranging error. Determine the location of each sensor relative to a designated origin.

The model

For this discussion, a “device” is a wireless sensor network node. Two devices are understood to be “neighbors” if they can communicate directly over the wireless channel. A “cluster” is a subset of the network whose devices are all neighbors.

The foundation of the algorithm is the “robust quadrilateral” - a collection of four neighboring devices, whose communication links are modeled as straight lines. These mutual links form three triangles and one quadrilateral. A “robust” triangle is one whose smallest interior angle has a specified minimum size. If all three triangles are robust, then the quadrilateral is robust.

The virtue of being robust is in minimizing ambiguity inherent in determining location based on range measurements [3]. *Figure 3* demonstrates that different network graphs may possess exactly the same edge lengths. With a lower bound on the smallest angle among the transmission lines between the nodes of the quadrilateral, the probability of range measurement noise causing a node to be located on the wrong side of a flex line is reduced.

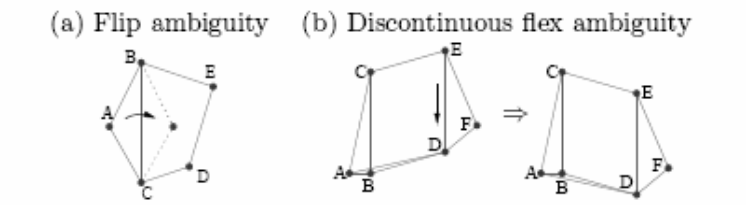

Figure 3: (a) Flip ambiguity. Vertex A can be reflected across the line connecting B and C with no change in the distance constraints. (b) Discontinuous flex ambiguity. If edge AD is removed, then reinserted, the graph can flex in the direction of the arrow, taking on a different configuration but exactly preserving all distance constraints.

With robust quadrilaterals identified in the network, a meta-graph may be constructed using robust quads as vertices. This "overlap graph" has an edge for each instance of two vertices sharing three common devices. The overlap graph (*Figure 6*) can then be used to determine the relative locations of each device in the network [3].

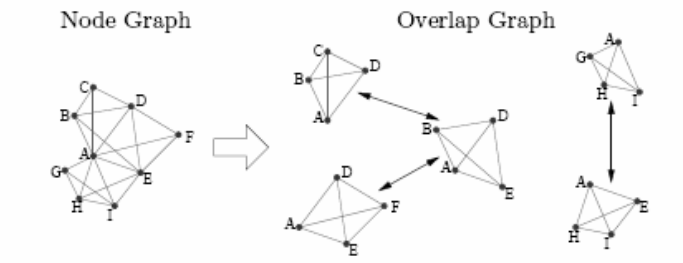

Figure 6: The duality between a cluster rooted at A and a graph of robust quads, which we call an *overlap graph*. In the overlap graph, each robust quadrilateral is a vertex. Edges are present between two quads whenever they share three nodes. Thus, if all four node positions are known for some quad, any neighboring quad in the overlap graph can use the three common nodes to trilaterate the position of the unknown node. A breadth-first search into the overlap graph from some starting quad, trilaterating along the way, localizes the cluster as described by Algorithm 2. Note that the overlap graph for a cluster can have distinct, unconnected subgraphs as shown in this example. Nodes that are unique to one subgraph cannot be localized with respect to those of an unconnected subgraph.

The Algorithm

For a given cluster, each device belonging to one or more robust quads can be localized with respect to a single reference device. This is accomplished by a sequence of two algorithms [3] (and construction of the overlap graph). The first algorithm determines all robust quads in the cluster. The second computes the positions of each robust-quad-contained device in the cluster.

Algorithm 1 Finds the set of robust quadrilaterals that contain an origin node i. Each quad is stored as a 4-tuple of its vertices and is returned in the set $Quads_i$. We assume that distance measurements have already been gathered as follows: $Meas_j$ is a set of ordered pairs (k, d_{jk}) that represent the distance from node j to node k. $d_{\min}$ is the robustness threshold, computed from the measurement noise.

```
1: for all pairs (j, d_ij) in Meas_i do
2:   for all pairs (k, d_jk) in Meas_j do
3:     Remove (j, d_kj) from Meas_k
4:     for all pairs (l, d_kl) in Meas_k do
5:       for all pairs (m, d_lm) in Meas_l do
6:         if m ≠ j then
7:           continue
8:         Retrieve (k, d_ik) from Meas_i
9:         Retrieve (l, d_il) from Meas_i
10:        if IsRobust(d_jk, d_kl, d_lj, d_min) AND
           IsRobust(d_ij, d_ik, d_jk, d_min) AND
           IsRobust(d_ij, d_il, d_lj, d_min) AND
           IsRobust(d_ik, d_il, d_kl, d_min) then
11:          Add (i, j, k, l) to Quads_i
12:    Remove (k, d_jk) from Meas_j
```

Algorithm 2 Computes position estimates for the cluster centered at node i. This algorithm does a breadth-first search into each disconnected subgraph of the overlap graph created from $Quads_i$ and finds the most complete localization possible. At the end of this algorithm, $Locs_{best}$ is a set containing pairs $(j, \mathbf{p})$ where $\mathbf{p}$ is the estimate for the x-y position of node j. Any neighbors of i not present in $Locs_{best}$ were not localizable.

```
1: Locs_best := ∅
2: for each disconnected subgraph of the overlap graph do
3:   Locs := ∅
4:   Choose a quad from the overlap graph.
5:   p0 := (0, 0) {Position of the origin node}
6:   p1 := (d_ab, 0) {First neighbor sets x-axis}
7:   α := (d_ab^2 + d_ac^2 − d_bc^2) / (2 d_ab d_ac)
8:   p2 := (d_ac α, d_ac √(1 − α^2)) {Localize the second neighbor relative to the first}
9:   Add (a, p0), (b, p1), and (c, p2) to Locs
10:  for each vertex visited in a breadth-first search into the overlap graph do
11:    if the current quadrilateral contains a node j that has not been localized yet then
12:      Let pa, pb, pc be the x-y positions of the three previously localized nodes.
13:      p' := Trilaterate(pa, d_aj, pb, d_bj, pc, d_cj)
14:      Add (j, p') to Locs
15:  if length(Locs) > length(Locs_best) then
16:    Locs_best := Locs
```

Methods

My objective was to develop working code that could demonstrate the operation of the algorithm on a data set representing a small number of wireless nodes. A class and dataflow diagram of the system is shown (*Figure 7*). Each Device object has the potential to regard itself as the origin node of its cluster.

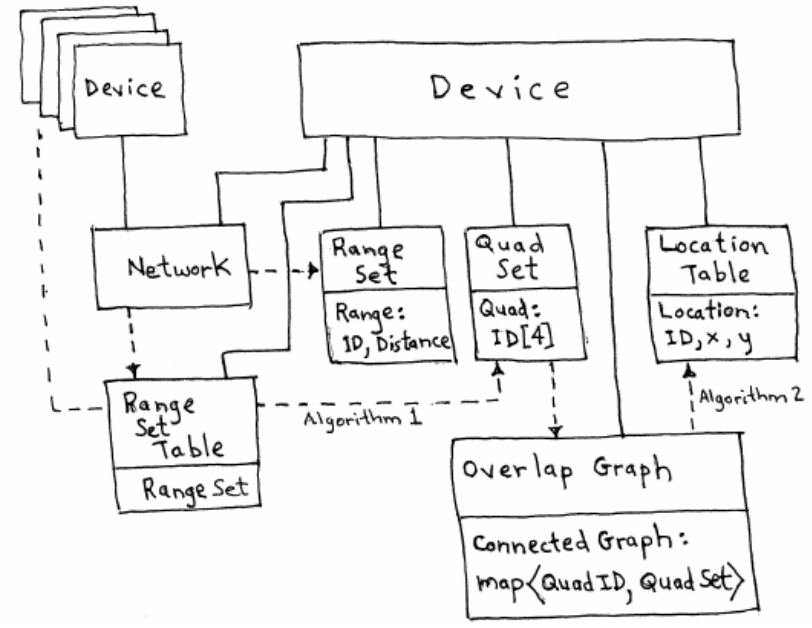

Figure 7

From its Network interface, the Device obtains a set of range measurements. Each Device must be aware not only of its own RangeSet, but also of the RangeSets of all other Devices in its cluster.

The output of Algorithm One is a QuadSet: a collection of Quad objects, each of which is merely a list of Device ID's. Then the OverlapGraph must be formed (the second step in the process). The OverlapGraph serves as input to Algorithm Two (the third step in the process). The output from Algorithm Two is the table of relative locations for each qualified Device in the cluster.

The Device class public method localize() implements the three-step process. It calls the methods buildQuads() (Algorithm One) and buildOverlapGraph(), then implements Algorithm Two by iterating through the connected component(s) of the OverlapGraph in a breadth-first search (*Appendix – Device.api.h, Device.h, and Device.cpp*).

buildQuads() uses a private constant method isRobust(const double, const double, const double) that returns a boolean answer to the question, Do the three supplied edge distances represent a robust triangle? This method uses the Law of Cosines to determine each interior angle, and compares the smallest to a threshold set according to desired noise performance (*Figure 8*). The design tradeoff is between localizing a larger proportion of the network nodes (using a smaller value of c) and achieving a lower probability of noise-induced localization error (using a larger value of c).

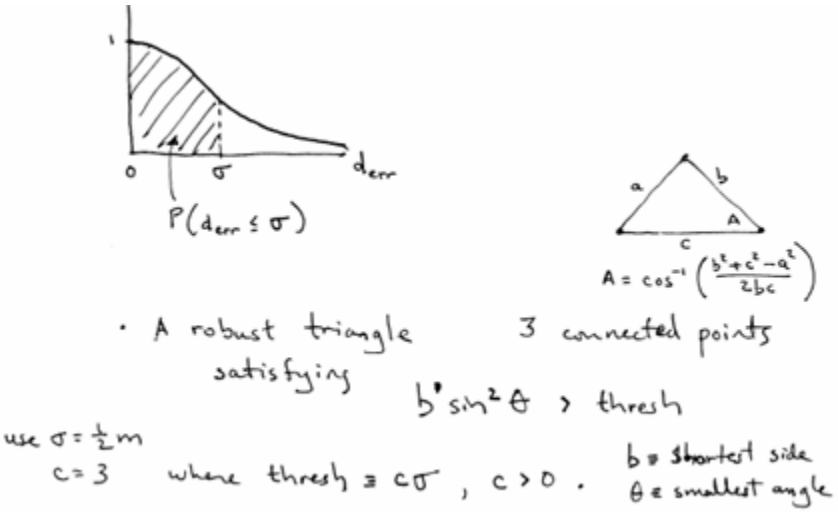

Figure 8

Results

I succeeded with Algorithm One, using a single Quad (*Figure 9*).

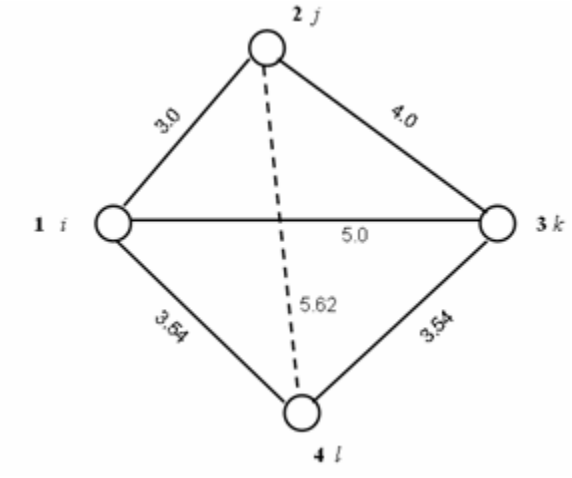

Figure 9

For my project the RangeSets for all cluster Devices were provided by a text file, shown below for a single Quad. The file format provides for listing the network ID of each cluster Device, followed by the list of its range measurements. Thus, the below text file represents a RangeSetTable object. (In practice each Device would obtain its own RangeSet, then would transmit its range data to all others in the cluster.)

```
h000000000001
h000000000002 3000.0
h000000000003 5000.0
h000000000004 3535.53
h000000000002
h000000000001 3000.0
h000000000003 4000.0
h000000000004 5622.5
h000000000003
h000000000001 2500.0
h000000000002 4000.0
h000000000004 3535.53
h000000000004
h000000000001 3535.53
h000000000002 5622.5
h000000000003 3535.53
#
```

The text file output from Algorithm One is shown below. It represents a series of overloaded '<<' operators belonging to the Device, Network, RangeSetTable, and QuadSet classes. The QuadSet listing at the end shows that the method buildQuads() correctly found and named the one robust quad in this cluster.

```
DEVICE ID: h000000000001
Threshold constant: 3
Noise sigma: 0.5

NETWORK:
Hardware address: h000000000001
Noise tolerance: 0.1

Device target:
c:\Documents and Settings\Owner\My Documents\School\CES 512\Project\
h000000000001.txt

Network target:
c:\Documents and Settings\Owner\My Documents\School\CES 512\Project\
network.txt

LOCAL RANGE SET:
h000000000002  3000

h000000000003  5000

h000000000004  3535.53


CLUSTER RANGE SETS:

Range Set for Device h000000000001
h000000000002  3000

h000000000003  5000

h000000000004  3535.53


Range Set for Device h000000000002
h000000000001  3000

h000000000003  4000

h000000000004  5622.5


Range Set for Device h000000000003
h000000000001  2500

h000000000002  4000

h000000000004  3535.53
```

```
Range Set for Device h000000000004
h000000000001  3535.53

h000000000002  5622.5

h000000000003  3535.53


QUAD SET:
QuadID:
h000000000001_h000000000002_h000000000003_h000000000004

Color = 0

Predecessor:
None
```

Summary

In one simple test case, Algorithm One was shown to identify a robust quadrilateral. The next step would be to build the simple overlap graph and implement Algorithm Two for determining the locations of three nodes relative to one. This would set the stage for further testing on more complex data sets. At some point, transition to an emulator platform or a TinyOS implementation might be practical.

Use of an object-based design aided me in translating the abstraction of Algorithm One as presented in the paper – which takes some confusing turns and in which not all steps are shown. It is not clear that there is overall merit in this approach, though. Perhaps too much time was spent on building and testing class features.

References

[1] Sahni S, Xu X. “Algorithms For Wireless Sensor Networks”. University of Florida. http://www.cise.ufl.edu/~sahni/papers/sensors.pdf

[2] Culler D, Estrin D, Srivastava M. “Overview of Sensor Networks”. IEEE Computer 37:8, Special Issue in Sensor Networks, Aug 2004. pp.41-49.

[3] Moore D, Leonard J, Rus D, Teller S. “Robust Distributed Network Localization with Noisy Range Measurements”. MIT. http://cgr.csail.mit.edu/netloc/sensys04.pdf