Using Clustering Information for Sensor Network Localization⋆

Haowen Chan, Mark Luk, and Adrian Perrig

Carnegie Mellon University
{haowenchan,mluk,perrig}@cmu.edu

Abstract. Sensor network localization continues to be an important research challenge. The goal of localization is to assign geographic coordinates to each node in the sensor network. Localization schemes for sensor network systems should work with inexpensive off-the-shelf hardware, scale to large networks, and also achieve good accuracy in the presence of irregularities and obstacles in the deployment area.

We present a novel approach for localization that can satisfy all the above desired properties. Recent developments in sensor network clustering algorithms have resulted in distributed algorithms that produce highly regular clusters. We propose to make use of this regularity to inform our localization algorithm. The main advantages of our approach are that our protocol requires only three randomly-placed nodes that know their geographic coordinates, and does not require any ranging or positioning equipment (i.e., no signal strength measurement, ultrasound ranging, or directional antennas are needed). So far, only the DV-Hop localization mechanism worked with the same assumptions [1]. We show that our proposed approach may outperform DV-Hop in certain scenarios, in particular when there exist large, well-spaced obstacles in the deployment field, or when the deployment area is free of obstacles but the number of anchors are very limited.

1 Introduction

Many wireless sensor network applications require information about the geographic location of each sensor node. Besides the typical application of correlating sensor readings with physical locations, approximate geographical localization is also needed for many sensor network applications such as location-aided routing [2], geographic routing [3], geographic routing with imprecise geographic coordinates [4, 5], geographic hash tables [6], and for many data aggregation applications.

Manually recording and entering the positions of each sensor node is impractical for very large sensor networks. To address the problem of assigning an approximate geographic coordinate to each sensor node, many automated localization algorithms have been developed. To obtain the information required for node locations, researchers proposed many approaches that make different assumptions: (1) quantitative ranging/directionality measurements [7–11]; (2) long range beacons [12–16]; (3) centralized processing [17, 18]; and (4) a flat, unobstructed deployment area. We do

⋆ This research was supported in part by CyLab at Carnegie Mellon under grant DAAD19-02-1-0389 from the Army Research Office, and grant CAREER CNS-0347807 from NSF, and by a gift from Bosch. The views and conclusions contained in this paper are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of Bosch, Carnegie Mellon University, NSF, the Army Research Office, the U.S. Government or any of its agencies.

not discuss protocols related to cases (1)–(3) because we are steering away from such assumptions.

Algorithms that assume a flat, unobstructed deployment area experience serious degradation in their position estimates in the presence of large obstacles and other irregularities in the deployment area. Most of the localization algorithms in current literature have been evaluated only for deployments clear of obstacles. However, such ideal deployments only represent the special case, while large obstacles are common in realistic settings. For a localization protocol to be practical, it is essential that it functions even in the presence of such irregularities. As can be seen in Figure 7 of Section 4, our algorithm has much better accuracy in recreating the topology of irregular deployments.

Generally, any algorithm that uses triangulation based on distance estimates to known *anchors* falls victim to errors caused by obstacles. An ***anchor*** is defined as a node that is aware of its own location, either through GPS or manual preprogramming during deployment. An example of a distance-triangulation protocol is the Ad-hoc Positioning System (APS) described by Niculescu and Nate [1]. They describe three methods of performing the distance estimate, the most widely cited of which is the DV-Hop method. DV-Hop uses a technique based on distance vector routing. Each node keeps the minimum number of hops to each anchor, and uses the hop count as an estimate of physical distance. Once a node has the estimated distance and location of 3 or more anchors, it performs least-squares error triangulation to estimate its own position. Nagpal et al. [19] describe a similar scheme but improve the accuracy of the distance estimation by using the average hop count of all the neighbors of a node as a distance estimate.

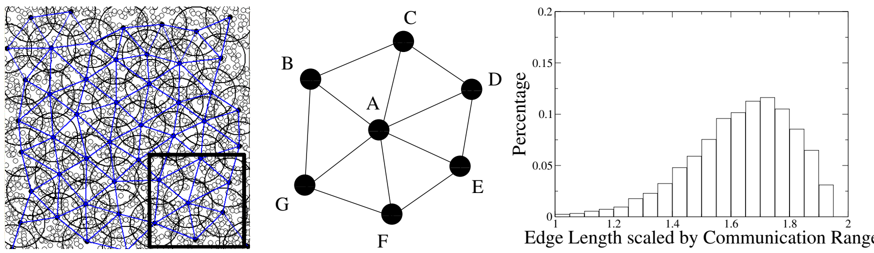
We present a localization scheme that requires no ranging or measuring equipment, no long range beacons, and no centralized processing, and is able to operate with arbitarily positioned anchor nodes. Furthermore, unlike DV-Hop, it makes no assumptions about the shape or internal topology of a deployment area: in particular, when the deployment area is occupied by large, well-spaced obstacles, our scheme significantly outperforms DV-Hop since it is able to re-create the physical topology of the network where DV-Hop cannot. Our scheme is based on the novel approach of first performing *sensor node clustering* on the the network in order to create a regular structure of representative nodes (called *cluster-heads*). To the best of our knowledge, this is the first localization protocol that does not make any assumption on the sensor node's hardware, yet performs well in certain classes of irregular topologies.

2 Sensor Network Clustering

2.1 Clustering Goals

Performing clustering on a sensor network deployment prior to localization has two advantages. First, it creates a regular pattern from which location information can be extracted. Second, it helps reduce the amount of communication overhead since only the cluster-heads need to be involved in the initial phase of the localization.

When a sensor network is first deployed, we cannot assume any regularity in the spacing or the pattern of the sensor nodes. Figure 1(a) shows that after clustering, the cluster-heads are regularly spaced throughout the network. We call two cluster-heads ***adjacent*** if there exists some other sensor node that is within communication range of both cluster-heads. Figure 1(a) shows the graph created by setting each cluster-head as



(a) After clustering (b) Expanded view (c) Distribution of edge lengths

Fig. 1. Effects of clustering.

a vertex and connecting each pair of adjacent cluster-heads with an edge. We call this graph the ***cluster-adjacency graph***. The edges in the graph are called ***cluster-adjacency edges***. The cluster-adjacency graph is mesh-like and has few edges that cross over each other. Furthermore, the variance in the length of the edges between adjacent nodes is small. Figure 1(c) is a histogram of edge length. Most edges fall within 1.3 to 1.7 r, with r being the maximum communication range between two nodes. The average edge length is 1.63 r and variance is 0.0309 r. Hence, the cluster-adjacency graph forms a regular structure from which location information can be extracted. This regularity is degraded slightly when the communication pattern is nonuniform (i.e. not a unit disk), or when the node density is low, but in general the regularity of cluster-head separation is always much greater than the distribution of physical distances between unclustered sensor nodes.

2.2 Modifying the ACE Algorithm

We chose to modify ACE [20] for use in our localization algorithm, because ACE already produces clusters with highly regular separation. Our localization techniques are not confined to the clusters produced by ACE; we can also use any other clustering technique that produces clusters with highly regular separation, such as the algorithm proposed by Younis and Fahmy [21].

A brief description of ACE is as follows (for details, please refer to the original paper [20]). The algorithm proceeds in a fixed number of iterations. In each iteration, sensor nodes gradually *self-elect* to become cluster-heads if they detect that many nodes in their neighborhood do not belong to any cluster. To achieve regular separation, these clusters then *migrate* away from other clusters by re-selecting their respective cluster-heads.

We modified ACE to improve the regularity of the separation between cluster-heads. First, we increased the number of iterations for ACE from 3 to 5, trading off increased communication cost for increased regularity. We also modified the migratory mechanism by approximating a *spring effect* between adjacent clusters. This effect causes clusters that are close together to migrate apart, and clusters that are too far apart to be attracted together. During the migration phase, each cluster-head evaluates the potential *fitness score* of each candidate node C in its neighborhood. The score for each candidate C is calculated as the total number of sensor nodes belonging to the cluster if C was to become the next cluster-head, plus a modifier for each adjacent cluster of C. Let s be the estimated separation between C and the adjacent cluster-head in terms of maximum communication radius r. s can be estimated by counting the number of common nodes in both the clusters - the greater the number of common nodes, the closer the separation of the two clusters. The final modifier is calculated via the function $g(s)$:

$$g(s) = \begin{cases} d(3(0.125 - (\frac{1}{2}(s-1.5)^2))) & \text{if } s \geq 1.5 \\ d(12(0.125 - (\frac{1}{2}(s-1.5)^2))) - 1.125 & \text{if } s < 1.5 \end{cases}$$

Where d is the total number of nodes in the neighborhood of C. The constants used above are empirically derived – their exact values are not important to the correctness of the algorithm. Note that the function above reaches a maximum at $s = 1.5$, meaning that clusters that are estimated to be more than $1.5r$ apart will be attracted together, while clusters that are less than $1.5r$ apart will be repelled from each other.

3 Localization Procedure

In this section we describe how the localization algorithm proceeds after clustering is complete. We first describe a naive version of our general approach using several strong assumptions. In subsequent subsections, we will eliminate these assumptions and improve the accuracy of our basic algorithm using more complex approaches.

3.1 The Basic Cluster Localization Algorithm

In this section we describe a high level overview of our general approach. Each of the steps described in this section are naive approaches which will be significantly improved in the subsequent sections.

Locally-Aware Anchors The algorithm starts from the anchor nodes, which are themselves cluster-heads and have knowledge of their geographical positions. We assume that these are ***Locally-Aware Anchor Nodes***, able to determine the geographical positions of all the cluster-heads adjacent to themselves. This could be performed by installing ranging and direction-finding hardware on the anchor nodes, or more practically by pre-selecting the cluster-head nodes in their neighborhood and directly programming these coordinates into the anchors. This increases the hardware or installation overhead of the scheme, hence we will eventually eliminate this assumption in Section 3.4.

Expanding the Calibrated Set The anchor nodes and their adjacent cluster-heads form an initial set of ***calibrated nodes*** which are aware of their positions. Given this base set of calibrated nodes, our algorithm will continually expand this set until all cluster-heads in the network have been calibrated. This is performed in a distributed manner where each cluster-head calibrates itself if two or more of its adjacent cluster-heads have successfully calibrated.

The self-calibration procedure uses the regularity of edge-lengths between cluster-heads to perform a position estimate. As an example, Figure 1(b) shows node A along with its adjacent neighbors, or its ***cluster-head neighborhood***. If node A knows the topological configuration of its cluster-head neighborhood as well as the estimated physical positions of two neighbors, C and D, A can estimate its own position by assuming some pre-determined standard value l for the length of the edges AC and AD. After A is calibrated, node B can similarly estimate its position based on positions of

C and A, further enabling node G to calibrate, and so on. In this manner, the set of calibrated nodes grows until all the cluster-heads in the network are calibrated. We present an significantly improved method for position estimation in Section 3.3.

Refining the Position Estimate The initial position estimate is performed based on the early position estimates of two neighbors. As more information is computed in the algorithm, more cluster-heads will be able to estimate their positions, and some already-calibrated cluster-heads may further improve their position estimates. Each cluster-head reacts to this new information by recomputing their own position estimates. A simple way of improving the position estimate would be to repeat the initial position estimation once for each adjacent pair of calibrated cluster-heads in the calibrating node's neighborhood, and then taking the average position of these results. To prevent the propagation through the network of very small changes, each node does not rebroadcast its updated position unless its difference from the previous position is larger than some threshold. When this occurs, we call it a *major* position update.

To improve the accuracy of this step, we have developed a more sophisticated algorithm for performing position refinement which will be presented in Section 3.2.

Termination Each node continues refining its position until either of two conditions occurs:

– *The node has reached some maximum number of major position updates*. We count the number of major position updates, and when it reaches 10 in the case of repeated initial calibration (see Section 3.3) or 60 in the case of mesh relaxation (see Section 3.2) then the node terminates and accepts its current estimated position as its final position.
– *The node has gone for some maximum time without receiving any updated information from its neighbors*. The amount of time to wait is chosen to be equal to the maximum time that position information needs to disseminate over the network, which is proportional to the diameter of the network. If the node has not received any new updates within this time frame after the last update, then there cannot be any further updates remaining in the system, and so the node terminates the algorithm.

Calibration of Follower Nodes Thus far, locality calibration has only been performed on the cluster-heads. When the cluster-heads have been fully localized, there remains the final step of calibrating the non-cluster-head nodes (i.e., the *follower* nodes. Various methods exist for calibrating these nodes. In our algorithm, each node takes the average of the estimated positions of all the calibrated nodes within its communication range (including cluster-heads and other follower nodes). This produces an localization accuracy for the non-cluster-head nodes that is very close to the localization accuracy of the cluster-heads. When this step is complete, all the sensor nodes are now localized.

3.2 Improved Position Refinement: Mesh Relaxation

We note that the goal of our algorithm is to solve for the geographical configuration of cluster-heads that is most likely, given the adjacency information of all cluster-heads and the position information of the anchors. We can approximate this solution by distributedly solving for the global configuration in which the square of the difference of the length of each edge from the known average edge length l is minimized.

To solve this problem, we use *mesh relaxation*, an approximation algorithm for finding the least-squares solution to a set of pairwise constraints. Mesh relaxation has previously been studied for localization in robotics [22–24]. A general description of mesh relaxation is beyond the scope of this paper; we describe how the method is applied to our localization protocol.

Each cluster-head is modeled as a mass point, and the distance between each pair of adjacent cluster-heads is modeled as a spring of length equal to the average edge length l. The calculation thus becomes equivalent to a physical simulation. Consider a cluster-head A. It has some estimated coordinate $p_{A,t}$ at time t, and we wish to continue the simulation to update its position in time $t + \delta_t$. Let the set of A's adjacent cluster-heads be S. Each of the members of S will exert a force on A. According to Hooke's law, this force can be expressed as $F = k\Delta x$, with Δx defined as the displacement of the spring from the spring's equilibrium length (set at the average edge length l), and k is the spring constant. Note that the value of k is irrelevant in this computation since we are looking for the point where all forces are equalized, which would be the same for any value of k. Hence, we let $k = 1$ and drop it from the equation. The resultant force on A at time t is:

$$\boldsymbol{F}_A = \sum_{B \in S} (|\boldsymbol{d}_{B,A}| - l)\hat{d}_{B,A}$$

The variable $\boldsymbol{d}_{B,A}$ represents the 2-dimensional vector of the separation between the estimated positions of cluster-heads B from A at time t, i.e. $\boldsymbol{d}_{B,A} = p_{B,t} - p_{A,t}$. The variable $\hat{d}_{B,A}$ represents the unit vector in the direction of B from A. l is the known average link length. We displace the position of A by $q\boldsymbol{F}_A$, a quantity proportional to the resultant force on A. Hence, the updated position of A is $p_{A,t+1} = p_{A,t} + q\boldsymbol{F}_A$. We iterate this process until the change in position is less than some threshold c.

The above algorithm is naturally parallelizable onto the cluster-heads; each cluster-head A calculates the forces acting on itself based on the current estimated locations of the nodes in its cluster-head neighborhood S and updates its own estimated position, which is then sent as an update to all the members of S.

3.3 Improved Initial Calibration

While mesh relaxation produces accurate localization results, if it begins with a poor initial estimated position, it takes many iterations to converge. Hence, having an initial estimate with high accuracy is essential to produce a workable algorithm. In this section we describe how to accurately make an initial estimate of a cluster-head's position based on the structure of the cluster-heads around it.

This algorithm consists of three steps. First, we acquire knowledge of a node's two-hop neighborhood to produce an ordered circular list of its adjacent cluster-heads. This list corresponds to either a counter-clockwise or a clockwise traversal of the set of adjacent cluster-heads. Then, we augment this list with some heuristic information about the relative separations of each cluster-head from its predecessor and successor in the circular list. Finally, the node calculates an estimate for its own position when two or more of its neighbors are calibrated.

Ordering the adjacent cluster-heads To orient a given cluster-head A correctly within the topology, we need to extract an ordering in its set of adjacent cluster-heads S. This ordering will produce information that we will later use to derive a location estimate for A based on the location estimates of the members of S.

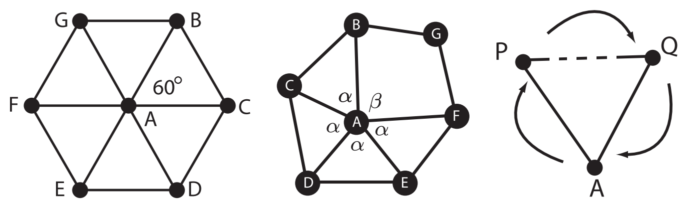
Figure 2(b) shows the neighborhood of cluster-head A. It has 5 adjacent cluster-heads, $\{B, C, D, E, F\}$. At the beginning of the protocol, A is aware of its neighborhood set (e.g. $\{D, C, F, B, E\}$) but not its order. The objective of this step of the algorithm is to derive an ordering on the set that corresponds to either a clockwise or counterclockwise traversal of the set, e.g., either (B, C, D, E, F) or (F, E, D, C, B). Note that the ordering is on a circle hence any cyclic shift of a correct sequence is still correct, e.g. (D, E, F, B, C).

We now introduce some terminology. Let the cluster-head for which we wish to derive the ordered circular list be the ***calibrating node***. The ***cluster-head neighborhood*** of a cluster-head is the set of cluster-heads that are *adjacent* to it (recall that two cluster-heads are considered adjacent if there exists some node which is in communication range of both of them). If two members of the cluster-head neighborhood of the calibrating node are also adjacent, then we call them ***directly linked*** with respect to the calibrating node. Examples of directly linked neighbors of the calibrating node A in Figure 2(b) are B and C, or C and D. If two members of the cluster-head neighborhood are not adjacent, but they are adjacent to another cluster-head that is not the calibrating node, nor in the cluster-head neighborhood, then we call them ***indirectly linked*** with respect to the calibrating node. For example, in Figure 2(b), B and F are indirectly linked with respect to A since they are both adjacent to G. Finally, if two members of the cluster-head neighborhood have no adjacent cluster-heads in common besides the calibrating node, then they are ***unlinked*** with respect to the calibrating node. Unlinked pairs are also called ***gaps*** since they represent a discontinuity in the cluster-head neighborhood of a node.

At the beginning of the algorithm, each cluster-head communicates its neighborhood information to all the members of its cluster-head neighborhood. Thus every cluster-head is aware of its cluster-head topology up to two edges away. If the cluster-adjacency graph has no cross edges in its physical embedding, then it is straightforward to construct the ordered circular list of neighbors for a calibrating node. The calibrating node selects any neighbor as a starting point and traverses the set of neighbors by selecting the next neighbor that has a direct or indirect link to the current node, then appending it to the list. The selected neighbor then becomes the current node and the process is iterated until the traversal returns to the starting point, or the current node has no direct or indirect links that have not already been traversed. In the latter case, traversal is restarted from the starting point in the opposite direction, and the nodes visited are pre-pended to the sequence. Ambiguities may arise if there are edges that cross over each other, since in this case there may be more than one possible choice for the next node in the traversal. However this occurs sufficiently infrequently that this choice can be resolved easily. For example, we use the heuristic of choosing the node that has the most common neighbors with the calibrating node, and skipping over any alternative nodes (i.e. not including them in the traversal at all).

At the end of this step, we have constructed an ordered circular list of the cluster-head neighborhood of a calibrating node. This list represents an initial estimate of the local physical topology of cluster-heads around the calibrating node.

Assigning angles to adjacent cluster-heads The next step in obtaining a physical mapping of this topology is to assign angular separations between subsequent members of the circular list. If all the cluster-heads in the list are all directly linked, we simply as-



(a) (b) (c)

Fig. 2. a) Assignment of angular shares when there are 6 adjacent cluster-heads all directly linked. The cluster-heads are assumed to be equally distributed, hence each angle is $360°/6 = 60°$.
b) Assignment of 5 angular shares when there are 4 direct links and 1 indirect links. Angles opposite to a direct link is denotes as α, and β represents the angle opposite to an indirect link.
c) Orienting A's circular list direction. If A is clockwise from Q with respect to P, and P is clockwise from A with respect to Q, then Q must be clockwise from P with respect to A.

sign equal angular shares to each sector. For example, in Figure 2(a), each sector is given $60°$ for a total of $360°$.

If there are one or more indirect links, we wish to assign larger angular shares to sectors subtended by indirect links since indirect links are usually longer than direct links. Letting the length of a direct link be l, we estimate the length of an indirect link as $\sqrt{2}l$ based on the intuition that the vertices forming an indirect link form a quadrilateral with the typical shape of a square. Hence, if α is the angle assigned to a direct link, then we should assign approximately $\beta = \sqrt{2}\alpha$ to an indirect link. Figure 2(b) shows an example of a node having 5 neighbors where there are 4 direct links and 1 indirect link, resulting in $\alpha = 66.5°$ assigned to the direct sectors and $\beta = 94.0°$ assigned to the indirect links.

In the case where there is a gap in the circular list (e.g. at the edge of the deployment area or next to an obstacle), we use the heuristic of assigning $\alpha = 60°$ to angles subtended by direct links and $\beta = 90°$ to angles subtended by indirect links.

When this portion of the algorithm is completed, every node has an ordered circular list representing its cluster-head neighborhood, as well as an angle between each adjacent pair of members of the circular list, representing the estimated angular separation of the pair.

Performing the position estimate At this point, the calibrating node is able to perform an initial position estimate if two or more of the nodes in its cluster-head neighborhood have already performed an initial position estimate. We call nodes which have successfully performed an initial position estimate, ***calibrated nodes***. A calibrated node not only has a position estimate but also has its circular list of its cluster-head neighborhood ordered in the canonical direction (i.e., physically clockwise or counterclockwise). For simplicity, we shall assume the canonical ordering is clockwise.

We first describe the algorithm using two calibrated nodes as reference nodes. Suppose the calibrating node A has two calibrated nodes P and Q in its cluster-head neighborhood. The first step in calibration is to orient A's circular list in the canonical clockwise direction. P and Q transmit to A their estimated positions (x_P, y_P) and (x_Q, y_Q) as well as their ordered cluster-head neighborhood list. These lists are ordered in the canonical clockwise direction. A observes its own position in these lists and deduces the ordering of its own list as follows. Figure 2(c) provides an illustration of the process.

Suppose in the list of P, A occurs after Q. Furthermore, in the list of Q, P occurs after A. Hence we know that A is clockwise from Q with respect to P, and P is clockwise from A with respect to Q, hence it must be that Q is clockwise from P with respect to A. Hence, in the ordered list of A, if the angular displacement of Q from P is greater than $180°$, then A needs to reverse its ordered list to put it in a clockwise order. The other case (where P is clockwise from Q with respect to A) follows an analogous argument.

Once A has determined the canonically correct ordering of its cluster-head neighborhood, it is now aware of which side of the line PQ it belongs. Hence, its initial position estimate can be calculated using basic trigonometry from the positions of P and Q and their estimated angular separation with respect to A. An estimate can be computed in several ways. We describe the method that we chose. The angle PAQ is known due to our angular assignment. Assuming that $AP = AQ$, we derive the angle QPA. Given this angle and the estimated position of P and Q, we can compute the angular bearing of A from P. We the compute A's estimated position with respect to P by assuming A's displacement from P is the known average edge length l. If there are multiple neighbors with known coordinates, we perform these operations once for each of them, i.e. for each P_i, compute the angular bearing of A from P_i and estimate A's position as a displacement of l along that bearing. After each estimate is computed, the final estimated position is calculated as the average (centroid) of all the estimates.

Repeated Initial Calibration We have found that this empirical process of estimating position is highly accurate. In fact, we can use this algorithm for both the initial position calibration, and for position refinement instead of performing mesh relaxation. When new information arrives as neighboring nodes update their positions, we merely perform the same position estimation algorithm again to obtain the new estimate. This process achieves comparable performance with mesh relaxation while incurring less communication overhead.

3.4 Self-orienting Anchors

Thus far, we have assumed that the anchors are "locally aware" (i.e. know the physical locations of all cluster-heads in their neighborhood) and that all nodes are aware of the average edge length between any two adjacent cluster-heads. We now describe an optimisation to remove these assumptions.

In this scheme, each anchor picks an arbitrary orientation and sets the average edge length l to 1. It assigns estimated positions to all the cluster-heads in its neighborhood according to the angular share system described in Section 3.3. Calibration then proceeds with respect to each anchor as normal. When calibration is complete, each cluster-head has formed a location estimate with respect to each anchor's arbitrary coordinate system. Specifically, each anchor is now calibrated with respect to every other anchor's coordinate system. All the anchors exchange this information along with their known physical coordinates.

Now each anchor can proceed to orient and scale its coordinate system to best fit the estimated positions of every other anchor under its coordinate system with its known physical location. Specifically, consider some anchor A. Number the other anchors $1..m$. After all calibration is complete, each of the other anchors sends to A their respective estimated locations $e_1, e_2, ..., e_m$ under A's arbitrarily chosen coordinate system. Each of the other anchors also sends to A their respective actual physical locations, i.e. $p_1, p_2, ..., p_m$. Now, A finds a transform T characterised by a rotation θ, a scaling factor c, and a bit r indicating whether or not reflection is needed, such that T is the transform that yields the lowest sum of squared errors between Te_i and p_i for each of the other anchors:

$$T = \underset{G}{\operatorname{argmin}} \sum_{i=1}^{m} (p_i - Ge_i)^2$$

At least 2 other anchors are needed to uniquely determine T. Once T is determined, it is then flooded to the rest of the network to allow the other cluster-heads to convert their estimated positions under A's coordinate system to actual physical locations.

This procedure will result in each cluster-head having several estimates of its position, one for each anchor. Based on the observation that position estimates increase in error with increasing hop distance from the anchor, each cluster-head uses the estimate associated with the closest anchor (in terms of cluster-head hop-count) and discards the others.

4 Results

Based on various combinations of the optimizations described in Section 3, we implemented three versions of our algorithm with various trade-offs:

1. Locally-Aware Anchors with Repeated Initial Calibration
2. Locally-Aware Anchors with Mesh Relaxation
3. Self-Orienting Anchors with Repeated Initial Calibration

With Locally-Aware Anchors, anchors are assumed to know the geographic positions of their immediate cluster-head neighborhood. This involves greater hardware or set-up cost. Self-Orienting Anchors do not make this assumption, and are only assumed to know their own geographic positions. The trade-off for removing this assumption is slightly lower accuracy and a higher communication cost.

In Repeated Initial Calibration, nodes are first calibrated using the method described in Section 3.3. When new information arrives and the nodes need to update their position estimates, they simply perform the initial calibration algorithm again to compute their new position. In Mesh Relaxation, the nodes are initialized similarly (i.e. using the technique of Section 3.3). However, as new information gets updated in the network, the nodes update their positions using mesh relaxation as described in Section 3.2. The trade-off is that mesh relaxation is more accurate than repeated initial calibration when using locally-aware anchors, but mesh relaxation requires more communication and takes a longer time. We used standard 32-bit floating point numbers during the simulated calculations, but we expect our results to hold also with lower levels of precision or with fixed-point computations.

We did not investigate the performance of self-orienting anchors with mesh relaxation, since these two methods did not interact well together and resulted in both higher communication overhead and less accuracy than self-orienting anchors with repeated initial calibration.

We provide a detailed quantitative analysis of each of the three versions of our algorithm. We evaluate our algorithms against the DV-Hop localization algorithm [1] with the smoothing optimization described by Nagpal et al. [19], because this is the only algorithm that also assumes no ranging/directional measurements, no long-range beacons, and no centralized processing. We investigated the performance of DV-Hop using normal anchors, as well as with Locally-Aware anchors for some scenarios. As we show from our results, although DV-Hop often has better accuracy in deployment settings with no obstacles and many well-placed anchors, our algorithms often outperform DV-Hop in more realistic settings in the presence of obstacles, irregularities and randomly placed anchors.

4.1 Base Simulation Assumptions and Parameters

Our base simulation setup is described for reference; how we vary the parameters of this base setup will be described later in each set of results. To evaluate the algorithms, we set up experiments using a deployment of 10,000 nodes over a square region of $20r \times 20r$ where r is the maximum communication radius. We do not assume that nodes are synchronized in time; nodes would periodically run an iteration of the algorithm regardless of the state of its neighbors. Anchor nodes are distributed randomly throughout the deployment. The base setup does not include obstacles.

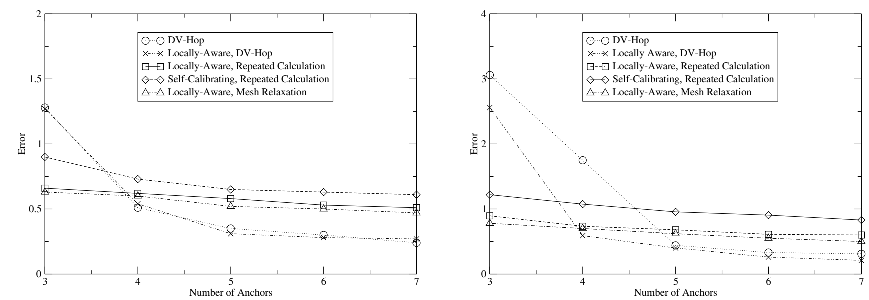
We used an irregular disc communication model. In order to simulate irregular communication range, we used the DOI model (or Degree of Irregularity) described by He et al. [14]. The transmission range of a node is a random walk around the disc, bounded by the maximum range r_{max} and minimum range r_{min}. We chose to set $r_{min} = 0.5r_{max}$. Let the range of a node in the bearing θ (in degrees) be r_θ. We start with $r_0 = 0.5(r_{min} + r_{max})$ and compute each subsequent r_θ as a random walk, i.e. $r_\theta = r_{\theta-1} + X(r_{max} - r_{min})D$ where X is a random real value uniformly chosen in the range $[-1, 1]$ and D is the degree of irregularity (DOI). Note that r_{theta} is not allowed to exceed r_{max} or go below r_{min}. r_{theta} represents only the transmission range of a node; since our schemes require bidirectional communication, we require both nodes to be within each other's respective transmission ranges in order to be able to communicate.

The metric for localization is the accuracy of the estimated position, which is measured as the distance between a node's estimated position and its true location, divided by maximum communication range r. The accuracy of a particular trial is measured as the average error over all nodes in the deployment. We also measured how much the average error varies among different trials.

4.2 Varying Number of Anchor Nodes in Uniform deployment

We varied the number of anchors from 3 to 7 to observe how accuracy is improved with increasing number of anchors for each algorithm. We also studied how much the error varies over different trials.

Figure 3 shows the average error for all four of our algorithms as well as DV-Hop with normal anchors and with locally aware anchors. The average localization error for all algorithms improves as the number of anchors increases. However, while the performance of our algorithms remained relatively stable as the number of anchors were varied, DV-Hop showed a very high sensitivity to the number of anchors. With the minimum number of anchors (3), regardless of whether these anchors were locally aware, DV-Hop typically incurs higher error than any of our algorithms. With 7 anchors, however, DV-Hop's average error improves to roughly half of ours. This suggests that DV-Hops requires significantly more anchors than our protocols in order to be maximally effective. Furthermore, the rapid degradation of the performance of DV-Hops

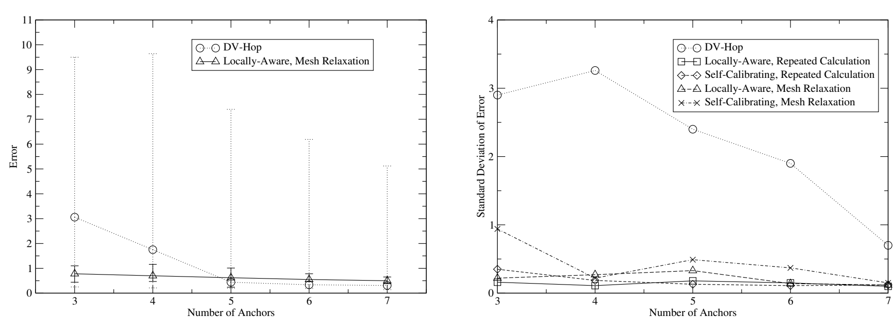


(a) DOI=0.05 (b) DOI=0.20

Fig. 3. Average Error with Varying Number of Anchors

as the number of anchors decreases indicates that it is not robust in scenarios where anchor node failure could be a factor. On the other hand, our algorithms provide uniformly good performance even with a very small number of anchor nodes, indicating dependable performance even if anchor nodes are subject to failure.

Figure 3 also shows that our self-orienting algorithms exhibit slightly poorer but comparable performance to the algorithms with locally aware anchors. This is a indication that the algorithms for self-orientation are reasonably effective.



(a) Min and max avg. error (b) Std Dev of avg. error

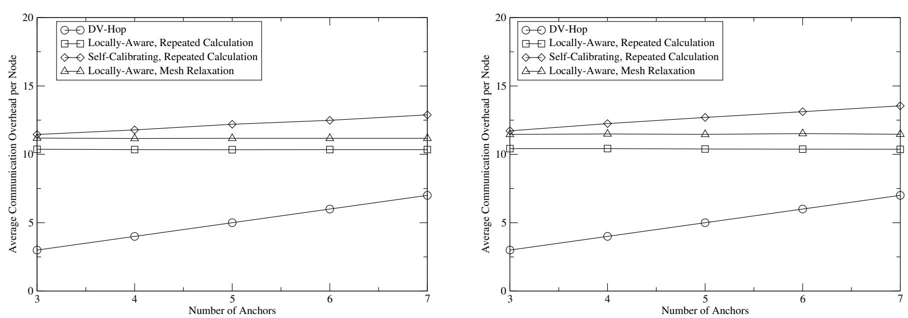
Fig. 4. Spread of Average Error for the different schemes, DOI=0.2

Figure 4a shows how the accuracy varies throughout different trials, with the error bars representing the minimum and maximum average error among all the trials. For clarity only one of our algorithms is shown; the other three exhibit similar behavior. Figure 4b graphs the standard deviation of the average error of each scheme over all our trials. We observe that although the expected error of DV-Hop becomes lower than our algorithms when five or more anchors are used, the variance is always much higher. In the best case, DV-Hop generates extremely accurate estimates. However, DV-Hop's error in the worst case scenario is significantly higher than the worst case error of our algorithm.

The intuition is that DV-Hop algorithm is more sensitive to the relative placement of anchors since it uses triangulation from anchors to estimate each node's position. Triangulation provides highly inaccurate results when anchors are placed in a co-linear fashion or are too close together. Clustering, on the other hand, has the advantage of not being significantly effected by positioning of anchor nodes. In certain cases, anchor nodes placed near each other actually improves performance. Random placement of anchor nodes thus proves to be a great advantage of cluster localization. As sensor networks become commodity technologies, random placement of anchors will be desirable because it allows for deployments by untrained personnel, instead of needing a specialized engineer to plan the process.

4.3 Communication Overhead

We measured the communication overhead of each of our schemes. We note that the overhead of performing clustering formed the bulk of our communications cost. Since the subsequent localization only involves cluster-heads, the communication cost for the network is low. An average of 9.11 communications per node were required for our modified ACE, while the localization schemes at most about 3 more communications per node on top of that. Hence, all our schemes achieve comparable communication costs to DV-Hop.



(a) DOI=0.05 (b) DOI=0.20

Fig. 5. Communication Overhead

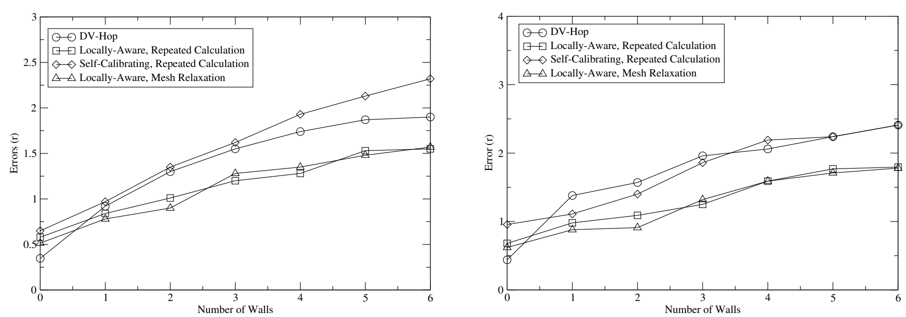
4.4 Obstacles

In this section, we study deployments with obstacles. Previously, we showed that DV-Hop is often more accurate in the case where obstacles are absent from the deployment field and numerous anchors are present. However, in more realistic scenarios, obstacles of various size and shape can disrupt communication and consequently interfere with localization.

Our 2 types of obstacles are *walls* and *voids*. Walls are represented as a line segment with length of 250 units, or half the length of the deployment field. Walls can be oriented in any direction, and all communication through the wall is blocked. Voids are areas of various fixed shapes that are off-limits during deployment. Our experiments investigated the effect of irregularly placed walls and regularly-spaced voids on the various schemes.

Since DV-Hop counts the number of hops between nodes to estimate distance, it almost always overestimates distances when the 2 nodes are separated by some type of obstacle. This is because DV-Hop uses hop count as an estimator of physical distance. When an obstacle is between an anchor and a calibrating node, hop counts can be inflated leading to a large overestimate of the physical distance. This can negatively affect the accuracy of the position estimation.

Clustering, on the other hand, is not significantly effected by obstacles. As shown in Figure 7(c) , the regular structure of cluster-heads are preserved around obstacles. Thus, localization based on clustering typically has much better performance than DV-Hops when faced with obstacles.

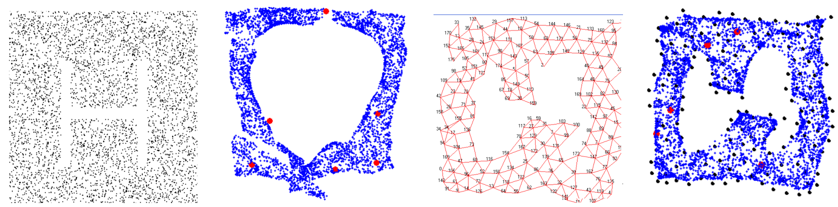


(a) DOI=0.05

(b) DOI=0.20

Fig. 6. Accuracy with obstacles, 5 anchor nodes

Figure 6 plots the accuracy of localization while increasing number of walls when using 5 anchor nodes. Locally-Aware schemes had the best performance, consistently outperforming DV-Hop whenever there are walls in the deployment area. The Self-Orienting schemes did not do as well in this scenario, yielding slightly worse performance compared with DV-Hop. In performing this series of tests, we observed the main flaw of clustering localization: our schemes perform by creating a map of the deployment area at the cluster-head level, which represents a relatively coarse granularity of resolution (about $1.5r$). Hence, our schemes perform best when here is sufficient space between obstacles to allow the regular but coarse structure of the clustering mesh to pass through the gap. If obstacles are placed close together (e.g. a gap of less than one communication radius), then the structure of cluster-heads through the gap may be too coarse to allow cluster-head localization to traverse the gap, leading to a failure in localization for certain segments of the deployment area. The self-calibrating scheme is particularly vulnerable to this effect since if an anchor is unable to find the estimated positions of at least two other anchors, it is unable to calibrate its own relative coordinate space and is thus almost useless for localization. Because the walls used in this experiment are extremely long and placed independently at random, their placement would occaisionally create small gaps though which DV-Hop could perform localization but our cluster-based schemes could not. This explains the relatively small degree of the performance improvement of our schemes over DV-Hop in these scenarios.



(a) Actual (b) DV-hop (c) Clustering (d) Localized

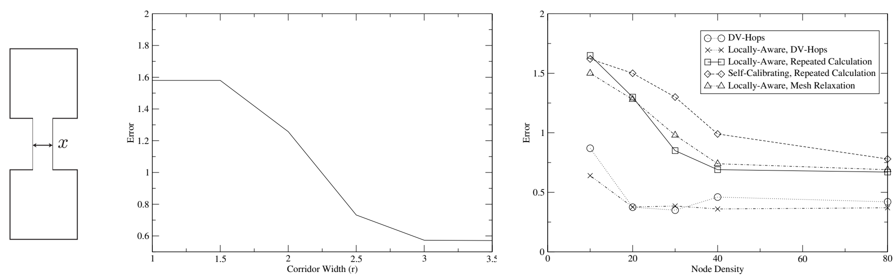
Fig. 7. Large Obstacles

	Cross	H	S	Thin S
DV-Hop	0.768	1.903	4.328	3.670
Locally Aware DV-Hop	0.686	1.650	5.145	3.575
Locally Aware, Mesh Relaxation	0.685	1.228	1.316	1.443
Locally Aware, R.I.C.	0.969	1.177	1.699	1.459
Self-Orienting, R.I.C.	1.235	1.469	2.951	2.919

Table 1. Average error for selected irregular deployments (5 anchors, DOI=0.05)

Table 1 shows the average error for selected deployments of *voids*, which are more well-spaced obstacles which were designed with sufficient gaps between obstacle features to enable the coarse-grained clustering structure to pass through and map out the entire deployment area. The name of each deployment represents what type of void is in the deployment field. For example, cross is a large void in the middle of the deployment field in the shape of a cross. H is an obstacle in the shape of a large capital H as represented in Figure 7 and S is a similar large obstacle constructed in the shape of an S using straight line segments. Figure 7(a) shows the actual deployment of nodes for obstacle H. Figure 7(b) shows how DV-Hop is unable to reconstruct occluded areas. Figure 7(c) shows that our cluster-head localization phase yields a good reconstruction of the deployment area which leads to good localization accuracy for all nodes as shown in Figure 7(d).

The experimental results confirm our hypothesis that for well-spaced, deeply concave obstacles, clustering localization always performs significantly better than DV-Hop, much greater than the improvement shown in Figure 6 where the obstacles were not well-spaced. The Cross-shaped obstacle was sufficiently convex in shape that DV-Hop retained relatively good accuracy and performed roughly as well as cluster localization. However, both the H and S deployments possess deep cul-de-sacs which could not be accurately triangulated by DV-Hop (see Figure 7(b)). However, our clustering methods allowed our schemes to reconstruct the shape of the deployment area (see Figure 7(c)) which yielded significantly better accuracies. The self-orienting schemes suffered inaccuracies since occaisionally some anchors were unable to deduce estimated positions of at least two other anchors and were thus unable to self-calibrate. However, this is not a fundamental weakness of clustering localization and can probably be addressed by more sophisticated self-calibration algorithms.



(a) (b) Corridor Experiment (c) Node Density

Fig. 8. Limitations of Cluster-Localization

We hypothesize that the accuracy of cluster-localization is dependent on having sufficient clearance space between obstacles to allow the regular cluster-head mesh structure to pass through. To investigate the extent of this effect, we simulated our self-calibrating scheme in a deliberately anomalous "dumbbell-shaped" deployment consisting of two large square deployment areas joined by a long narrow corridor (see Figure 8(a)). Only two anchors were placed in each each main deployment area; thus cluster-localization can only be successful if sufficient information can pass through the corridor to allow calibration of each anchor (unsuccessful nodes simply adopt the location of the nearest anchor). The results are shown in Figure 8(b) for a DOI of 0.2. As can be seen, accuracy degrades significantly when the corridor is too narrow to fit sufficient cluster-heads for the reconstruction of the cluster-head topology; however once the corridor is sufficiently wide ($2.5r$ in this case), the cluster-head structure is able to reconstruct the shape of the corridor and yields highly accurate results.

We note that our schemes only yield high accuracy for sufficiently dense networks; Figure 8(c) shows that at low densities (less than 30 nodes per circle of radius r), the accuracy of cluster localization suffers while DV-Hop retains its good performance. This is because at low densities, clustering is not as tight, and hence the number of adjacent cluster-heads is lower, leading to a more sparse cluster-head topology which yields less information for localization.

5 Conclusion

Localization continues to be an important challenge in today's sensor networks. In this paper, we propose to use clustering as a basis for determining the position information of sensor nodes. To the best of our knowledge, this is the first paper to consider this approach. Our clustering-based approach has many benefits: it is fully distributed, it provides good accuracy, it only requires that three randomly placed sensor nodes know their geographic position information, and it works with standard sensor node hardware without requiring any special hardware such as ultrasound or other ranging equipment. Moreover, our approach provides accurate position information even in topologies with walls and other concave structures, as long as the granularity of the obstacle features are on the same order as the separation between cluster-heads.

References

1. Niculescu, D., Nath, B.: Ad hoc positioning system (APS). In: IEEE GLOBECOM. (2001) 2926–2931
2. Ko, Y.B., Vaidya, N.: Location-aided routing (LAR) in mobile ad hoc networks. In: Proceedings of MobiCom, ACM (1998) 66–75
3. Karp, B., Kung, H.T.: GPSR: greedy perimeter stateless routing for wireless networks. In: Proceedings of MobiCom. (2000) 243–254
4. Newsome, J., Song, D.: GEM: Graph embedding for routing and data-centric storage in sensor networks without geographic information. In: Proceedings of SenSys. (2003) 76–88
5. Rao, A., Ratnasamy, S., Papadimitriou, C., Shenker, S., Stoica, I.: Geographic routing without location information. In: Proceedings of MobiCom. (2003) 96–108
6. Ratnasamy, S., Karp, B., Yin, L., Yu, F., Estrin, D., Govindan, R., Shenker, S.: GHT: a geographic hash table for data-centric storage. In: WSNA 2002. (2002)
7. Capkun, S., Hamdi, M., Hubaux, J.P.: Gps-free positioning in mobile ad-hoc networks. Cluster Computing **5** (2002)
8. Savarese, C., Rabaey, J., Langendoen, K.: Robust positioning algorithms for distributed ad-hoc wireless sensor networks. In: Proceedings of the General Track: USENIX Annual Technical Conference. (2002) 317–327
9. Savvides, A., Han, C., Srivastava, M.B.: Dynamic fine grained localization in ad-hoc sensor networks. In: Proceedings of MobiCom. (2001) 166–179
10. Savvides, A., Park, H., Srivastava, M.B.: The n-hop multilateration primitive for node localization problems. Mobile Networks and Applications **8** (2003) 443–451
11. Ji, X., Zha, H.: Sensor positioning in wireless ad-hoc sensor networks with multidimensional scaling. In: Proceedings of IEEE Infocom. (2004)
12. Bahl, P., Padmanabhan, V.: Radar: an in-building rf-based user location and tracking system. In: Proceedings of IEEE Infocom. (2000)
13. Bulusu, N., Heidemann, J., Estrin, D.: GPS-less low cost outdoor localization for very small devices. IEEE Personal Communications Magazine **7** (2000) 28–34
14. He, T., Huang, C., Blum, B., Stankovic, J.A., Abdelzaher, T.: Range-free localization schemes for large scale sensor networks. In: Proceedings of MobiCom. (2003) 81–95
15. Priyantha, N.B., Chakraborty, A., Balakrishnan, H.: The Cricket location-support system. In: Proceedings of MobiCom. (2000)
16. Nasipuri, A., Li, K.: A directionality based location discovery scheme for wireless sensor networks. In: Proceedings of WSNA. (2002) 105–111
17. Doherty, L., Pister, K.S.J., Ghaoui, L.E.: Convex position estimation in wireless sensor networks. In: Proceedings of IEEE Infocom). Volume 3. (2001) 1655 –1663
18. Shang, Y., Ruml, W., Zhang, Y., Fromherz, M.P.: Localization from mere connectivity. In: Proceedings of Mobihoc. (2003) 201–212
19. Nagpal, R., Shrobe, H., Bachrach, J.: Organizing a global coordinate system from local information on an ad hoc sensor network. In: Proceedings of IPSN. (2003)
20. Chan, H., Perrig, A.: ACE: An emergent algorithm for highly uniform cluster formation. In: Proceedings of EWSN. (2004)
21. Younis, O., Fahmy, S.: Distributed clustering in ad-hoc sensor networks: A hybrid, energy-efficient approach. In: Proceedings of IEEE INFOCOM. (2004)
22. Duckett, T., Marsland, S., Shapiro, J.: Learning globally consistent maps by relaxation. In: Proceedings of IEEE ICRA, San Francisco, CA (2000)
23. Golfarelli, M., Maio, D., Rizzi, S.: Elastic correction of dead-reckoning errors in map building. In: Proceedings of IEEE ICRA, B.C., Canada (1998)
24. Howard, A., Matarić, M., Sukhatme, G.: Relaxation on a mesh: a formalism for generalized localization. In: Proceedings of IROS, IEEE (2001)