Analysis of Distributed, Low-Power Algorithms for Localization in Sensor Networks
Jana van Greunen
EE 228 Walrand

Abstract
A set of requirements for successful localization in sensor networks is developed. These requirements and criteria are then used to evaluate and compare several existing localization algorithms. The following extensions to the algorithms are made: real-time error/accuracy estimation and lower-energy computation. Real-time error estimation is desirable when an application requires accuracy guarantees or for early termination of the algorithm if the current position estimates already meet the required accuracy. A distributed version of the Cramer-Rao Bound is used to provide an estimate of the minimum variance or error. The second extension involved the combination of two algorithms to utilize the iterative nature and multi-hop operation of one and the simple computation of the other. Simulations comparing the performance of the combined algorithms show that it performs well for sparse networks.

1. Introduction
A sensor network consists of many small, densely deployed sensor nodes. The sensor nodes communicate with each other through wireless (usually RF) transmission. Sensor networks are an emerging technology and there are many possible applications. For example, military applications such as distributed surveillance and targeting, real-time traffic monitoring, smart environment control, managing inventory, weather data gathering, and the monitoring of buildings during earthquakes.

The work in this paper is principally motivated by the Pico-radio project. Pico-radio is the Berkeley Wireless Center’s sensor network project with the two following main goals: develop meso-scale, low cost, self-powered radio nodes and to deploy these nodes in a self-configuring network. Thus, for Pico-radio, minimizing energy and power consumption is crucial. However, other sensor network implementations may have different requirements that lead to special design and algorithm constraints.

In general, to be effective, sensor networks need the following properties:

• ad-hoc organization
• self-configuring
• low-power
• low-cost
• fault tolerant

Many applications of sensor networks require that the nodes in the network are aware of their geographic location. Localization refers to the process by which the nodes discover this location. In this paper, several existing localization algorithms will be introduced. The goal of this project is to evaluate and extend these algorithms to incorporate new features such as current error estimation, early termination when error bounds are low, and better convergence guarantees.

This project report is organized as follows: Section 2 introduces the basic concepts of localization and criteria for evaluation. Section 3 is an overview of existing localization algorithms, followed by the evaluation and comparison of these algorithms in Section 4. Two extensions, error estimate and reduced computation, are explored in Section 5. Section 6 describes future work and Section 7 concludes this report.

2. Localization Overview

2.1 Localization Techniques
One solution to the problem of localization is to equip each node with a Global Positioning System (GPS) receiver. Unfortunately, this is a poor solution in terms of power consumption and per-unit cost. In addition, GPS does not perform well in buildings. A more practical solution is to have a few nodes that know their position a priori (these nodes are referred to as known/beacon/anchor nodes). The unknown nodes in the network can then calculate their positions by “measuring” (in terms of hops or distance) their respective distance from these nodes.

Localization in sensor networks poses several challenges. First, the complexity of the localization algorithm will often grow with the number of nodes in the sensor network. Thus, a scalable localization algorithm must be distributed. Second, the network has constrained communication and computation resources. A good algorithm should try to optimize for both these factors. Third, there is currently no way to accurately measure distance between nodes using RF transmission. Several methods have been devised (e.g., via received signal strength or time of arrival). Klemmer and Whitehouse [4] found that received signal strength was the most promising method, but even so, it produced
large errors on individual measurements and was only reliable once the network was trained for the characteristics of a particular environment. Consequently a solution has to be tolerant to large errors in the distance measurement.

2.2 Criteria for Algorithm Evaluation

Despite the implementation-specific constraints of a sensor network and the diverse requirements of each application running on the network, a set of global performance criteria can be devised. For specific applications, only a subset of the criteria may be relevant. Furthermore, there are often trade-offs between the criteria. The following criteria encompass both necessary and desirable properties of any localization algorithm:

- **Scalability.** A scalable algorithm minimizes the required per-node computation as the network size grows. Typically, scalability is accomplished by incorporating information from one-hop neighbors only or partitioning the network into smaller regions of communication.

- **Communication energy.** In general, it is desirable to keep the total energy spent on communication as low as possible to preserve the energy in the sensor network. However, certain applications may require higher accuracy, resulting in increased communication energy.

- **Computation energy.** It is similarly desirable to minimize total computation energy in the network. However, there is often a trade-off between computation energy, speed of convergence, and communication requirements.

- **Accuracy.** While it is impossible to estimate node locations with 100% correctness, some applications may require an upper bound on the estimation error.

- **Error tolerance.** The localization algorithm must tolerate large and sporadic range measurement errors. Range errors occur for two reasons: current ranging technology is erroneous and, fundamentally, there will always be interference in the environment causing non-line-of-sight errors.

- **Convergence speed.** Applications may need fast or bounded convergence times, for example a time-critical surveillance sensor network.

- **Good performance across networks.** The algorithm should perform well on different implementations of sensor networks. That is, the algorithm should not depend on a specific network topology or specialized hardware.

- **Estimate of error.** The algorithm should provide an estimate of the position accuracy at all times. This will facilitate an algorithm that can terminate early when the current error estimate is below some threshold.

3. Existing algorithms

This section provides an introduction to six existing localization algorithms, along with a discussion of their advantages, disadvantages, and energy costs. In the next section, each of these algorithms will be evaluated and compared against each other, and they serve as the basis for some proposed extensions and improvements.

3.1 Centralized LP

The centralized linear algorithm described in [3] uses connectivity constraints to solve the localization problem. This algorithm is based on the assumption that if two nodes can communicate with each other, they must lie within the communication radius $R$ of each other. This ability to communicate is mathematically equivalent to a 2-norm constraint on the node positions.

$$\|a - b\|_2 < R \Rightarrow \begin{bmatrix} I_2 R & a - b \\ (a - b)^T & R \end{bmatrix} \succeq 0 \quad (xR)$$

Figure 1: LMI constraint (Image taken from [3])

Note: the region of communication for a node must be a convex circle of radius $R$. This is not true for real RF transmissions.

The communication constraints of all the nodes can be combined to form a convex feasibility problem with the following mathematical formulation:

Minimize : $c^T x$

Subject to : $Ax < b$

Where $x$ is a vector of $(x, y)$ coordinates for each node in the network. A standard LP solver can solve this convex problem.

**Advantages**

This algorithm is theoretically well defined and easy to analyze because it can be formulated as LP problem. It is well suited to a heterogeneous sensor network where certain nodes have more computational power than others. The communication time from the central node to the furthest node in the sensor network (plus the computation time at the central node) determines the convergence time of this algorithm. This algorithm achieves the best possible accuracy because it incorporates all available inter-node information.
Disadvantages

The pooling of information toward a central point will deplete network resources around this point. Moreover, the algorithm is computation intensive because the computation involved in the linear program scales as the square size of the network. Thus the algorithm is not scalable and cannot easily be implemented on a low-power, low-cost node. In addition, the convex linear program fails when the regions of communication are not convex. The regions will not be convex when the range between two nodes is underestimated, so the algorithm is not tolerant to large range errors.

Cost

The following notation is used for all cost calculations in this paper: $D$ is the physical dimension of the problem (i.e. either 2D or 3D), $c$ is the average degree (number of neighbors) of nodes in the network. $N$ is the total number of nodes in the network and $A$ is the number of anchors in the network.

The average degree of each node and the average distance of nodes from the central node determine the communication cost for the centralized LP algorithm. Computation cost is determined by the number of matrix operations to solve the LP. For a $m \times n$ matrix the number of operations is roughly $O(mn^2 + n^3/3)$. In this problem $n = D \times N$ and $m$ is approximated by average degree $c$.

<table>
<thead>
<tr>
<th>Energy consumption</th>
<th>Total Computation $c(D \times N)^2 + (D \times N)^3/3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total Communication $pathlen \times cN$</td>
<td></td>
</tr>
</tbody>
</table>

3.2 Rectangular Intersection

This rectangular intersection algorithm [11] was developed as an extension to the centralized LP. It is assumed that all nodes are placed within a square region $S$. The region $S$ is then further subdivided into $n^2$ cells of area $(s/n)^2$. Instead of the “continuous” communication model used in [3], where the communication constraint is modeled as a circle of radius $R$, this approach uses the discrete communication model. In the discrete communication model, the communication radius is given in terms of a number of cells $P$. Also, the communication region is no longer a circle of radius $R$, it is now a square with side length $P$. For example, if node $k$ and $j$ are one-hop neighbors (nodes $j$ and $k$ can communicate with each other), $k$ lies within a convex square centered at $j$. With the discrete model, the combination of connectivity constraints reduces to calculating the intersection of squares (see figure 2).

Advantages

This algorithm requires very little computation and communication energy because nodes only communicate with their neighbors. The algorithm converges rapidly and completes in a single stage. The only delays are measuring single-hop distances and the minimal processing at each node. It can also be shown that as number known neighbors go to infinity, the expected rectangular intersection for an unknown node is accurate to a single cell. Due to its distributed nature, the algorithm is scalable. In fact, the complexity of the algorithm at each node is completely unrelated with the size of the network.

Disadvantages

The main drawback of this approach is that each unknown node needs to communicate with several known neighbors. Thus, if there are few known nodes in the network this algorithm performs very poorly. This al-

![Image: Intersection of rectangles (Image taken from [11])](image-url)
algorithm also relies on the assumption that the regions of communication are convex and will perform poorly in the presence of multi-path interference that can cause the radius of communication to deviate from $P$.

**Cost**

The computation consists of finding $D$ maxima and minima resulting from the intersections. For each intersection there are $c$ comparisons and $2c$ additions. Thus the total computation cost will be $3N \cdot D \cdot c$. On average, each node needs to communicate once with its $c$ neighbors for a total of $N \cdot c$ across the network.

<table>
<thead>
<tr>
<th>Energy consumption</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Total Computation</strong></td>
</tr>
<tr>
<td><strong>Total Communication</strong></td>
</tr>
</tbody>
</table>

### 3.3 Network Coordinate

The network coordinate algorithm, described in [2] is a two-phase algorithm. During the first phase, each node $i$ in the network measures the distances $d_j$ to all of its one-hop neighbors $j$, where $j \in K_i$. The distances between two neighbors $j$ and $k$ is also sent to node $i$. This information enables each node to place its one-hop neighbors in a local coordinate system.

![Figure 3](image3.png)

Figure 3: The node at the origin can locate $j$ and $q$ in its local coordinate system if it knows the distance between itself and $q$, as well as the distance between $q$ and $j$. (Image taken from [2])

When all nodes have constructed a local coordinate system, their coordinate systems can be aligned to a network coordinate system. Nodes $i$ and $k$ align their coordinate systems in a pair-wise fashion by rotation around a common neighbor $j$ (see figures 4 and 5).

### Advantages

This algorithm requires few known nodes. There is only moderate computation at each node and moderate amount of communication over time.

### Disadvantages

This algorithm has a very long convergence time, because the alignment of local coordinate systems has to propagate through the network. In addition, the rotation errors are also propagated through the network and will compound to give inaccurate results.

### Cost

To construct the local coordinate system a node communicates with each of its neighbors ($N \cdot c^2$ total). The same order of calculations is required to place each neighbor in the node’s local coordinate system. During the global coordinate construction, each node calculates a rotation and translation with at least one other node in the network.

<table>
<thead>
<tr>
<th>Energy consumption</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Total Computation</strong></td>
</tr>
<tr>
<td><strong>Total Communication</strong></td>
</tr>
</tbody>
</table>
3.4 DV-hop

This algorithm was developed in [6] to prevent propagation of error across multiple hops. Thus it computes the location of unknown nodes based on an “average” distance between nodes. The known nodes flood the network with their positions. Each unknown node stores a list of the positions of known nodes and the number of hops they are from these known nodes. Once an unknown node has its ‘hop-distance’ (number of hops-average distance) from more than three non-collinear known nodes it can compute its position by triangulation (solving $Ax=b$).

**Advantages**

This algorithm is fairly simple, easy to implement, and tolerant to range measurement errors. It takes a medium amount of time to converge because the hop information must propagate through the network. DV-hop does prevent errors from accumulating in the network. The algorithm is scalable because each node needs only information from at least three anchors.

**Disadvantages**

DV-hop can achieve only limited accuracy because of the averaging effect. This algorithm also has high computation cost due to the maximum likelihood triangulation. DV-Hop does not perform very well when the network topology is irregular and the actual distances deviate greatly from the average distances used in the algorithm.

**Cost**

To compute the triangulation, the hop information first needs to be linearized. This requires $D\times(A-1)$ subtractions. Forming $b$ (in $Ax=b$) requires $(A-1)\times(2\times D+1)$ additions and $2\times(A-1)\times(D+1)$ multiplications. Once the data has been linearized, the added complexity from the least-squares algorithm is $O((A-1)D^2 + D^3/3)$ [9]. The communication cost consists of flooding the hop counts through the network, thus it is $A\times N$.

<table>
<thead>
<tr>
<th>Energy consumption</th>
<th>$N\times(\beta D^2 + D^3/3)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total Computation</td>
<td>$A\times D^2 + D^3/3$</td>
</tr>
<tr>
<td>Total Communication</td>
<td>$A\times N$</td>
</tr>
</tbody>
</table>

3.5 Grid of Beacons

In this algorithm [1] the beacons or known nodes are placed on a regular grid. These known nodes periodically transmit their locations. An unknown node receives above $Cmthresh$ percentage of the transmitted messages the unknown node is ‘connected’ to the beacon. The unknown node then computes its position as the centroid of all the beacons that it is connected to.

$$ (X_m, Y_m) = \left( \frac{\sum_{i=1}^{k} X_i}{k}, \frac{\sum_{i=1}^{k} Y_i}{k} \right) $$

**Advantages**

This algorithm is similar to the rectangular intersection in that it relies only on information from immediate neighbors. Thus it is scalable, distributed, requires very low computation and communication energy, and it converges rapidly.

**Disadvantages**

In order to achieve reasonable accuracy, there needs to be a regular and dense grid of beacon nodes. This may increase the cost of the network and may be infeasible for under certain conditions when the positioning of beacon nodes cannot be controlled.

**Cost**

Each node computes a sum from $D\times A$ locations followed by one division. The average number of anchors that each node can communicate with ($A$) is less than $c$, the average degree of nodes. Thus the total computation in the network is $N$ times $D\times c$. Each node also only communicates with its beacon neighbors. Thus the total communication in the network is $N\times c$.

<table>
<thead>
<tr>
<th>Energy consumption</th>
<th>$D\times c \times N$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total Computation</td>
<td>$D\times c \times N$</td>
</tr>
<tr>
<td>Total Communication</td>
<td>$N\times c$</td>
</tr>
</tbody>
</table>

3.6 Start-up and refinement

This algorithm [10] consists of two stages: start-up and refinement. The start-up phase provides an initial estimate of each node’s position using DV-hop. During the following phase (refinement), the nodes try to improve their initial position estimate. The nodes achieve this improvement by measuring the distances to all of their one-hop neighbors and updating their own position accordingly. All positions in this algorithm are calculated with linearized maximum likelihood triangulation [9]. The refinement method improves the initial position estimate if the errors in the initial position estimates are uncorrelated.

Two additional measures are taken to improve the performance of the refinement stage. First, ill-connected nodes are prevented from participating. An ill-connected node is a node that does not have independent references, i.e. it may have more than three neighbors but receives hop-information from less than
three of these. Second, confidence estimates are used as weights during triangulation. All unknown nodes start with a confidence estimate of 0.1 while known node have a confidence of 1.0. When an unknown node updates its position estimation it also updates its confidence estimate to the average of its neighbors confidence estimates. Thus the average confidence in the network is expected to increase with the number of refinement iterations. The confidence estimates improve the convergence and accuracy of the refinement stage.

**Advantages**

Start-up and refinement achieves reasonable accuracy. The network can control the number of iterations - thus there is a potential for real-time optimization of energy spent versus accuracy achieved. This algorithm is fairly tolerant to range errors if the network connectivity is high. (High connectivity implies that unknown nodes have redundant information available.) Start-up and refinement is also a scalable algorithm because nodes mainly communicate only with one-hop neighbors.

**Disadvantages**

This algorithm is somewhat computation intensive due to the iterative triangulation. The algorithm may not converge to an accurate solution if the initial position estimations are very inaccurate or the errors are correlated. Also, depending on the topology of the network the algorithm may take a long time to converge.

**Cost**

The computation cost of this algorithm is very similar to that of DV-hop. The only difference is that in the refinement stage the least-squares triangulation is computed $s$ (# of iterations) times and the number of nodes is $c$, the average degree of nodes in the network. The initial communication cost is also the same as that of DV-hop.

<table>
<thead>
<tr>
<th>Energy consumption</th>
<th>Total Computation</th>
<th>$N \cdot (3D+1) \cdot (A + x) + O((A + s)D^2 + D^3/3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total Communication</td>
<td>$A \cdot N + c \cdot N \cdot s$</td>
<td></td>
</tr>
</tbody>
</table>

4. **Evaluation & Comparison**

The following section evaluates the algorithms according to the criteria outlined in Section 2. The most important criterion for localization is scalability. If an algorithm is not scalable it cannot be implemented on a real network. Except for the centralized LP algorithm, all algorithms presented in the previous section are scalable. For PicoRadio, cost is another important criterion; sensor networks must consist of cheap nodes. The rectangular intersection and grid algorithms require many nodes that know their own location, and will increase the cost of the network. The other algorithms do not place such constraints on the network.

Another important consideration is the accuracy of the algorithm. Error propagation is a problem in the network coordinate algorithm. When the local coordinate systems are aligned the errors compound resulting in large final error. On the other hand, the rectangular intersection and grid algorithms will only have good accuracy if there are many known nodes in the network. In contrast, DV-hop and start-up & refinement achieve moderate accuracy in a variety of network conditions. Refinement is more accurate than DV-hop because uses actual distance information to improve the initial estimates.

Finally, choosing an algorithm that minimizes the energy spent during the localization is desirable. The rectangular intersection and grid algorithms are the most energy efficient because they communicate only with neighbors and have very simple computation requirements. Network coordinate and start-up & refinement are moderately efficient because both algorithms mainly use information from immediate neighbors and have complex but limited size calculations. In contrast, the centralized LP is very inefficient because the linear program grows with $N^2$. For the purpose of PicoRadio, the start-up and refinement algorithm is the best. It achieves fair accuracy with moderate energy expenditure and supports real-time optimization of accuracy and energy by varying the number of refinement iterations. A summary of the above comparison is given in figure 6.

Some observations:

- All algorithms lack an estimation of position error.
- Most algorithms have either moderately high computation requirements and work with few known nodes, or they require many known nodes and have lower computation requirements.

Based on the above observations, the second part of this project extends existing algorithms to incorporate the following features: error estimates and reduced computation with few anchor nodes.
5. Extensions

This section presents two extensions to the localization algorithms presented above: accurate error estimation and reduced energy consumption with lower accuracy.

5.1 Accurate error estimation

Localization is an estimation problem in the presence of noise. There are two sources of noise. First there is noise in ranging measurements. Second, when an unknown node uses another unknown node k to estimate its position, the uncertainty in k’s position contributes additional noise. The aim of error estimation is quantify the error or variance of the new position estimate.

The Cramer-Rao Bound is an analytical tool that provides a lower bound on the covariance matrix of any unbiased estimator \( \hat{\theta} \). The Cramer-Rao Bound is the inverse of the Fisher information matrix.

Mathematical formulation [5]:

\[
\text{Var}(\hat{\theta}) \geq [F^{-1}(\theta)]_{ij}
\]

where \( F(\theta) \) is the Fisher information matrix given by:

\[
[F(\theta)]_{ij} = -E[\frac{\partial^2 \ln p(x;\theta)}{\partial \theta_i \partial \theta_j}]
\]

where \( p(x;\theta) \) is the conditional density function.

Patwari and Hero [7] use the Cramer-Rao Bound to calculate the minimum resulting variance for a position estimate made in the presence of ranging errors, the first type of noise. The following is a brief overview of their approach.

Assumptions

- \( d_{ij}^s \) (measured distance from node \( i \) to \( j \) for all \( i, j \)) are Gaussian distributed.
- The measurements \( d_{ij} \) are independent for all \( i, j \).
- The variance associated with the measured distance is independent of the size of the measurement and is the same for all measurements in the network.
- Communication is symmetric, i.e. if node \( i \) can communicate with \( j, j \) can also communicate with \( i \). Moreover, \( d_{ij} = d_{ji} \) for all \( i, j \).
- Use RSS to measure distance (this affects the exact shape of the density – results for TOA are very similar).

Notation

- \( H(k) \) – set of nodes with which node \( k \) has made measurements.
- \( d_{ij} \) is the shortest distance from point \( (x_0,y_0) \) to the line between the device \( i \) and device \( j \).
- \( \sigma^s \) variance in distance measurement.
Note: The Cramer-Rao Bound is only valid for an unbiased estimator. However, a biased estimator can asymptotically reach this bound. Patwari et al. also explores ways to reduce the estimator bias.

For this estimation problem, the Fisher information matrix naturally partitions into $x$ and $y$ blocks:

$$ F = \begin{bmatrix} F_{xx} & F_{xy} \\ F_{xy} & F_{yy} \end{bmatrix} $$

To find the matrix entries (the partial derivatives of the log of the joint density function) it is necessary to define the joint density function first. Using the independence assumption, the joint density can be written as a product of individual densities. When the logarithm is taken, this product becomes a sum of the following form (from [7]):

$$ l(\theta) = \sum_{i=1}^{N-1} \sum_{j \in H(i)} \left[ C_1 - \frac{b}{8} \left( \log \frac{d_{i,j}^2}{d_{i,j}^2} \right)^2 \right] $$

Now, considering measurements for a single unknown node $i$ surrounded by $N$ known nodes, and substituting $d_{i,j} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$, then taking the second order partial derivatives of $l(\theta)$ results in the following matrix entries:

$$(F_{xx})_{i,k} = \frac{b}{2} \sum_{j=1}^{N} \left( \frac{(x_i - x_j)^2}{(x_i - x_j)^2 + (y_i - y_j)^2} \right)^k, \quad k = 1 $$

$$(F_{xx})_{i,k} = \frac{b}{2} \sum_{j=1}^{N} \left( \frac{(x_i - x_j)^2}{(x_i - x_j)^2 + (y_i - y_j)^2} \right)^{(k-1)\text{ mod } 2}, \quad k \neq 1 $$

$$(F_{xy})_{i,k} = \frac{b}{2} \sum_{j=1}^{N} \left( \frac{(x_i - x_j)(y_i - y_j)}{(x_i - x_j)^2 + (y_i - y_j)^2} \right)^k, \quad k = 1 $$

$$(F_{xy})_{i,k} = \frac{b}{2} \sum_{j=1}^{N} \left( \frac{(x_i - x_j)(y_i - y_j)}{(x_i - x_j)^2 + (y_i - y_j)^2} \right)^{(k-1)\text{ mod } 2}, \quad k \neq 1 $$

To find the Cramer-Rao bound we take the expected value of the inverse of this matrix:

$$ \sigma \geq \frac{F_{xx} + F_{yy}}{F_{xx}F_{yy} - F_{xy}^2} = \frac{\sum_{i=1}^{N} d_{i,j}^{-2}}{b^2 \sum_{i=2}^{N-1} \sum_{j \neq i} d_{i,j}^{-2} \left( d_{i,j} \right)^{-2} d_{i,j}^{-2} } $$

In order to incorporate Patwari et al.’s approach in the start-up and refinement algorithm each node must calculate its own Cramer-Rao Bound for each update step in the algorithm. During the start-up phase the inter-node distance measurements are not used. Thus, it does not make sense to compute the Cramer-Rao Bound. The error in this phase is due to the deviation of nodes from their “average” distances. Therefore, the deviation can be captured in the position variance of nodes. The position variance can be estimated by calculating the variance of inter-node distances for a subset of nodes. This variance serves as an initial error estimate.

The Cramer-Rao Bound is applicable during the refinement phase. However, nodes will not only use distances from known nodes, but also from other unknown nodes. The uncertainty in the positions of these unknown nodes effectively increases the variance of their distance measurements. One way to incorporate this noise in the estimation is to increase $\sigma_{d_{i,j}}$ of each $d_{i,j}$ (inter-node distance measurement) by the current variance of node $j$’s position estimate. So the new variance $\sigma_{d_{i,j}} = \sigma_{d_{i,j}} + \sigma_j$. Increasing the variance accounts for the uncertainty in $j$’s position. Thus the Cramer-Rao Bound can provide an estimate for the minimum expected variance at each step in the start-up and refinement algorithm.

### 5.2 Low energy computation

As previously discussed, the start-up and refinement algorithm has some desirable properties: it is distributed, scalable, and supports real-time optimization. However, the computation of the maximum likelihood triangulation involves at least one matrix inversion (the dimension of the matrix will be $D \times \epsilon$ where $D$ is the dimension of the space and $\epsilon$ is the number of neighbors). Thus, the energy efficiency of the algorithm would improve if the computation can be simplified.

One way to simplify the computation is to substitute the triangulation, which tries to find the intersect of $\epsilon$ circles, by a computation that can find the intersection of $\epsilon$ rectangles. Thus, for the low-energy version of start-up and refinement, the triangulation is replaced by rectangular intersection.

The low-power algorithm offers significant computation savings, each node performs $O(\epsilon \times D)$ comparisons instead of $O(\epsilon \times D^2)$ multiplications and additions. Thus there is a computation reduction of $D^2$ and the nodes can have simpler hardware because there are no multiplications.

### 5.3 Simulation & Results

Simulations of the start-up and refinement algorithm were performed using the OMNET++ network simulation package developed by Andras Varga [12]. The test networks were created by randomly placing 400 nodes in $200^2$ rectangular area. The percentage of known nodes was varied from 2-20% across trials.
Similarly, the communication range was varied from 5-15 in order to control the average connectivity of nodes in the network. The simulated range measurements contained 5% error on average for all the simulations.

Figure 7 Average position error after start-up (with 5% range error) taken from [10]

In general, the accuracy of the algorithm increases as the connectivity and percentage of anchor nodes increase. This trend is expected because there is more redundant information available in the network and is observable in both algorithms. The low-power algorithm performs as good as the original algorithm at low connectivity levels and about 2 times worse at higher connectivity.

Figure 8 Average position error after rectangular start-up (with 5% range error)

Results are slightly different than expected for the refinement stage. At low connectivity, the algorithms perform comparably. However, the low-energy algorithm does not show the increase in accuracy with increasing connectivity that was evident in the original algorithm. This trend of increasing accuracy with connectivity in the original algorithm can be attributed to the use of weights. The weights prevent a triangulating node from relying heavily on information from neighbors with correlated errors. Because weights are not used in the rectangular intersection, the error correlation between neighbors (more evident in a high-connectivity network) prevents the algorithm from converging accurately. Thus, the low-energy method is appropriate only for the start-up phase or for sparse networks.

6. Future work

The current localization algorithms assume that all nodes in the network are identical. It would be interesting to study non-homogenous networks, in which some nodes are more powerful than others. In this case substantial improvements in accuracy may result from partial centralization or hierarchical subdivision of the algorithm among nodes. Also, it would be interesting to study the effects of a few long distance measurements to combat the error that arises across multiple hops.

7. Conclusion

In this paper, a set of requirements for localization in sensor networks was introduced. Several existing localization algorithms were presented and compared. From the comparison it is evident that the performance of distributed algorithms is heavily dependent on underlying network topology. There is also the problem of error compounding across multiple hops. This error propagation can be reduced by increased computation and calculation. This analysis motivated the following extensions: real-time error estimation and lower-energy computation for lower accuracy. The real-time error estimation is achieved by using the Cramer-Rao Bound.
Lower-energy computation is implemented by substituting rectangular intersection for triangulation in the start-up and refinement algorithm. Results show that the lower-energy algorithm performs acceptably in sparse networks (up to two times worse than the start-up and refinement algorithm) but is not suited for dense networks.

8. References


