# 1 Localization in Sensor Networks

Jonathan Bachrach and Christopher Taylor
Computer Science and Artificial Intelligence Laboratory
Massachusetts Institute of Technology
Cambridge, MA 02139
{taylorc, bachrach}@mit.edu

Location, Location, Location

—anonymous

## 1.1 INTRODUCTION

Advances in technology have made it possible to build ad hoc sensor networks using inexpensive nodes consisting of a low power processor, a modest amount of memory, a wireless network transceiver and a sensor board; a typical node is comparable in size to 2 AA batteries [10]. Many novel applications are emerging: habitat monitoring, smart building failure detection and reporting, and target tracking. In these applications it is necessary to accurately orient the nodes with respect to a global coordinate system in order to report data that is geographically meaningful. Furthermore, basic middle ware services such as routing often rely on location information (e.g., geographic routing).

Ad hoc sensor networks present novel tradeoffs in system design. On the one hand, the low cost of the nodes facilitates massive scale and highly parallel computation. On the other hand, each node is likely to have limited power, limited reliability, and only local communication with a modest number of neighbors. These application contexts and potential massive scale make it unrealistic to rely on careful placement or uniform arrangement of sensors. Rather than use globally accessible beacons or expensive GPS to localize each sensor, we would like the sensors to self-organize a coordinate system.

In this chapter, we review localization hardware, discuss issues in localization algorithm design, present the most important localization techniques, and finally suggest future directions in localization. The goal of this chapter is to outline the technical foundations of today's localization techniques and present the tradeoffs inherent in algorithm design. No specific algorithm is a clear favorite across the spectrum. For example, some algorithms rely on prepositioned nodes (section 1.2.1) while others are able to do without. Other algorithms require expensive hardware capabilities. Some algorithms need a way of performing off-line computation, while other algorithms are able to do all their calculations on the sensor nodes themselves. Localization is still an new and exciting field, with new algorithms, hardware, and applications being developed at a feverish pace; it is hard to say what techniques and hardware will be prevalent in the end.

## 1.2 LOCALIZATION HARDWARE

The localization problem gives rise to two important hardware problems. The first, the problem of defining a coordinate system, is covered in section 1.2.1. The second, which is the more technically challenging, is the problem of calculating the distance between sensors (the ranging problem), which is covered in the balance of section 1.2.

### 1.2.1 Anchor/Beacon nodes

The goal of localization is to determine the physical coordinates of a group of sensor nodes. These coordinates can be global, meaning they are aligned with some externally meaningful system like GPS, or relative, meaning that they are an arbitrary "rigid transformation" (rotation, reflection, translation) away from the global coordinate system.

Beacon nodes (also frequently called anchor nodes) are a necessary prerequisite to localizing a network in a global coordinate system. Beacon nodes are simply ordinary sensor nodes that know their global coordinates a priori. This knowledge could be hard coded, or acquired through some additional hardware like a GPS receiver. At a minimum, three non-collinear beacon nodes are required to define a global coordinate system in two dimensions. If three dimensional coordinates are required, then at least four non-coplanar beacons must be present.

Beacon nodes can be used in several ways. Some algorithms (e.g. MDS-MAP, section 1.4.2) localize nodes in an arbitrary relative coordinate system, then use a few beacon nodes to determine a rigid transformation of the relative coordinates into global coordinates (see appendix B). Other algorithms (e.g. APIT, section 1.5.4) use beacons throughout, using the positions of several beacons to "bootstrap" the global positions of non-beacon nodes.

Beacon placement can often have a significant impact on localization. Many groups have found that localization accuracy improves if beacons are placed in a convex hull around the network. Locating additional beacons in the center of the network is also helpful. In any event, there is considerable evidence that real improvements in localization can be obtained by planning beacon layout in the network.

The advantage of using beacons is obvious: the presence of several pre-localized nodes can greatly simplify the task of assigning coordinates to ordinary nodes. However, beacon nodes have inherent disadvantages. GPS receivers are expensive. They also cannot typically be used indoors, and can also be confused by tall buildings or other environmental obstacles. GPS receivers also consume significant battery power, which can be a problem for power-constrained sensor nodes. The alternative to GPS is pre-programming nodes with their locations, which can be impractical (for instance when deploying 10,000 nodes with 500 beacons) or even impossible (for instance when deploying nodes from an aircraft).

In short, beacons are necessary for localization, but their use does not come without cost.

The remainder of section 1.2 will focus on hardware methods of computing distance measurements between nearby sensor nodes (i.e. ranging).

### 1.2.2 Received Signal Strength Indication (RSSI)

In wireless sensor networks, every sensor has a radio. The question is: how can the radio help localize the network? There are two important techniques for using radio information to compute ranges. One of them, hop count, is discussed in section 1.2.3. The other, Received Signal Strength Indication (RSSI), is covered below.

In theory, the energy of a radio signal diminishes with the square of the distance from the signal's source. As a result, a node listening to a radio transmission should be able to use the strength of the received signal to calculate its distance from the transmitter. RSSI suggests an elegant solution to the hardware ranging problem: all sensor nodes are likely to have radios – why not use them to compute ranges for localization?

In practice, however, RSSI ranging measurements contain noise on the order of several meters [2]. This noise occurs because radio propagation tends to be highly non-uniform in real environments (see figure 1.1). For instance, radio propagates differently over asphalt than over grass. Physical obstacles such as walls, furniture, etc. reflect and absorb radio waves. As a result, distance predictions using signal strength have been unable to demonstrate the precision obtained by other ranging methods such as time difference of arrival (section 1.2.4).

However, RSSI has garnered new interest recently. More careful physical analysis of radio propagation may allow better use of RSSI data, as might better calibration of sensor radios. Whitehouse [32] did an extensive analysis of radio signal strength, which he was able to parlay into noticeable improvements in localization. Thus, it is quite possible that a more sophisticated use RSSI will eventually prove to be a superior ranging technology, from a

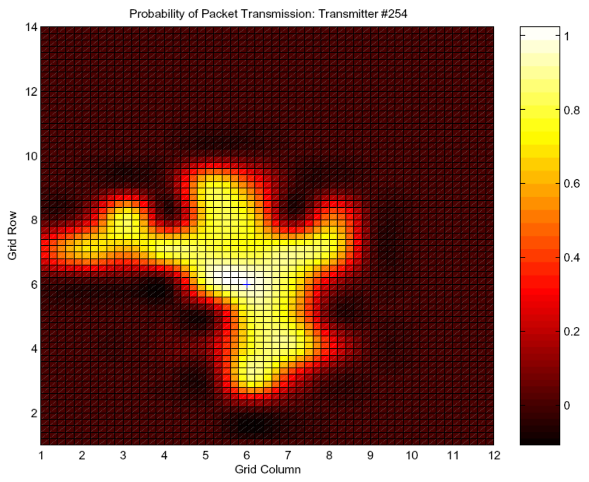

**Fig. 1.1** Diagram by Alec Woo, borrowed with permission from [32], which shows the probability of successful packet transmission with respect to distance from the source. It shows that the fixed-radius disk approximation of radio connectivity is quite inaccurate. It also demonstrates the difficulties inherent in retrieving distance information from signal strength.

price/performance standpoint. Nevertheless, the technology is not there today.

### 1.2.3 Radio Hop Count

Even though RSSI is too inaccurate for many applications, the radio can still be used to assist localization. The key observation is that if two nodes can communicate by radio, their distance from each other is less than $R$ with high probability, where $R$ is the maximum range of their radios, no matter what their signal strength reading is. Thus, simple connectivity data can be useful for localization purposes.

In particular, many groups have found "hop count" to be a useful way to compute inter-node distances. The local connectivity information provided by the radio defines an unweighted graph, where the vertices are sensor nodes, and edges represent direct radio links between nodes. The hop count $h_{ij}$ between sensor nodes $s_i$ and $s_j$ is then defined as the length of the shortest path in the graph between $s_i$ and $s_j$.

Naively, if the hop count between $s_i$ and $s_j$ is $h_{ij}$ then the distance between $s_i$ and $s_j$, $d_{ij}$, is less than $R * h_{ij}$, where $R$ is again the maximum radio range.

It turns out that a better estimate can be made if we know $n_{local}$, the expected number of neighbors per node. Then, as shown by Kleinrock and

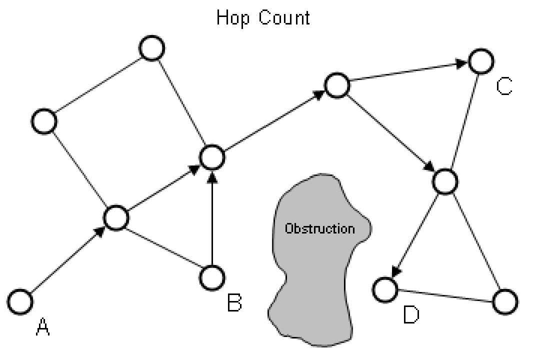

**Fig. 1.2** Examples of hop count. In this diagram, $h_{AC} = 4$. Unfortunately, $h_{BD}$ is also four, due to an obstruction in the topology. This is one of the ways that hop count distance metrics can experience dramatic error.

Silvester [14], it is possible to compute a better formula for the distance covered by one radio hop:

$$d_{hop} = R\left(1 + e^{-n_{local}} - \int_{-1}^{1} e^{-\frac{n_{local}}{\pi}(\arccos t - t\sqrt{1-t^2})} dt\right) \tag{1.1}$$

Then, $d_{ij} \approx h_{ij} * d_{hop}$. Experimentally[20], equation (1.1) has been shown to be quite accurate when $n_{local}$ grows above 5. However, when $n_{local} > 15$, $d_{hop}$ approaches $R$, so equation (1.1) becomes less useful.

There are two problems with using hop count as a measurement of distance. First, distance measurements are always integral multiples of $d_{hop}$. This inaccuracy corresponds to a total error of about $0.5R$ per measurement, which can be too high for some applications. Second, environmental obstacles can prevent edges from appearing in the connectivity graph that otherwise would be present. As a result, hop count based distances can be substantially too high, for example as in figure 1.2.

Nagpal et al [20] demonstrate algorithm that even better hop count distance estimates can be computed by averaging distances with neighbors. This benefit does not begin to appear until $n_{local} \geq 15$, however, it can reduce hop count error down to as little as $0.2R$.

### 1.2.4 Time Difference of Arrival (TDoA)

Time Difference of Arrival (TDoA) is a commonly used hardware ranging mechanism. In TDoA schemes, each node is equipped with a speaker and a microphone. Some systems use ultrasound while others use audible fre-

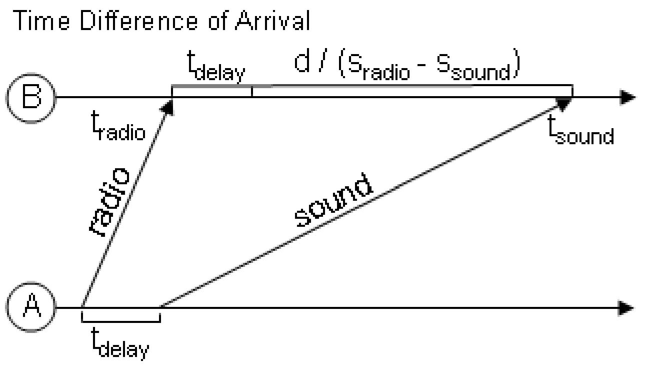

**Fig. 1.3** Time Difference of Arrival (TDoA) illustrated. Sensor A sends a radio pulse followed by an acoustic pulse. By determining the time difference between the arrival of the two pulses, sensor B can estimate its distance from A.

quencies. However, the general mathematical technique is independent of particular hardware.

In TDoA, the transmitter first sends a radio message. It waits some fixed interval of time, $t_{delay}$ (which might be zero), and then produces a fixed pattern of "chirps" on its speaker.

When listening nodes hear the radio signal, they note the current time, $t_{radio}$, then turn on their microphones. When their microphones detect the chirp pattern, they again note the current time, $t_{sound}$. Once they have $t_{radio}$, $t_{sound}$, and $t_{delay}$, the listeners can compute the distance $d$ between themselves and the transmitter using the fact that radio waves travel substantially faster than sound in air.

$$d = (s_{radio} - s_{sound}) * (t_{sound} - t_{radio} - t_{delay}) \tag{1.2}$$

TDoA methods are impressively accurate under line-of-sight conditions; however, they perform best in areas that are free of echoes, and when the speakers and microphones are calibrated to each other. Several groups are working to compensate for these issues, which will likely lead to even better field accuracy.

Nevertheless, rather good results can already be obtained, even in sub-par conditions. The Cricket ultrasound ranging system [3] can obtain close to centimeter accuracy without calibration over ranges of up to ten meters in indoor environments, provided the transmitter and receiver have line-of-sight.

The downside of TDoA systems is that they inevitably require special hardware to be built into sensor nodes, specifically a speaker and a microphone. TDoA systems perform best when they are calibrated properly, since speakers and microphones never have identical transmission and reception characteristics. Furthermore, the speed of sound in air varies with air temperature and humidity which introduces inaccuracy into equation 1.2. Finally, the line-of-sight constraint can be difficult to meet in some environments.

It is possible to use additional constraints to identify and prune bad ranging data ("outliers") [15]. Representative constraints include:

1. The range from node A to node B should be approximately equal to the range from node B to node A ($r_{AB} \approx r_{BA}$).
2. The pairwise ranges between nodes A, B, and C should obey the triangle inequality ($r_{AB} + r_{AC} \geq r_B C$)

In the end, many localization algorithms use time difference of arrival ranging simply because it is dramatically more accurate than radio-only methods. The actual reason why TDoA is more effective in practice than RSSI is due to the difference between using signal travel time and signal magnitude, where the former is vulnerable only to occlusion while the latter is vulnerable to both occlusion and multipath.

### 1.2.5 Angle of Arrival (AoA), Digital Compasses

Some algorithms depend on angle of arrival (AoA) data. This data is typically gathered using radio or microphone arrays, which allow a listening node to determine the direction of a transmitting node. It is also possible to gather AoA data from optical communication methods.

In these methods, several (3-4) spatially separated microphones hear a single transmitted signal. By analyzing the phase or time difference between the signal's arrival at different microphones, it is possible to discover the angle of arrival of the signal.

These methods can obtain accuracy to within a few degrees [25]. Unfortunately, angle-of-arrival hardware tends to be bulkier and more expensive than TDoA ranging hardware, since each node must have one speaker and several microphones. Furthermore, the need for spatial separation between speakers is difficult to accommodate as the form factor of sensors shrinks.

Angle of Arrival hardware is sometimes augmented with digital compasses. A digital compass simply indicates the global orientation of its node, which can be quite useful in conjunction with AoA information.

In practice, few sensor localization algorithms absolutely require Angle of Arrival information, though several are capable of using it when it is present.

## 1.3 ISSUES IN LOCALIZATION ALGORITHM DESIGN

### 1.3.1 Resource constraints

Sensor networks are typically quite resource-starved. Nodes have rather weak processors, making large computations infeasible. Moreover, sensor nodes are typically battery powered. This means communication, processing, and sensing actions are all expensive, since they actively reduce the lifespan of the node performing them.

Not only that, sensor networks are typically envisioned on a large scale, with hundreds or thousands of nodes in a typical deployment. This fact has two important consequences: nodes must be cheap to fabricate, and trivially easy to deploy. Nodes must be cheap, since fifty cents of additional cost per node translates to $500 for a one thousand node network. Deployment must be easy as well: thirty seconds of handling time per node to prepare for localization translates to over eight man-hours of work to deploy a 1000 node network.

Localization is necessary to many functions of a sensor network; however, it is not the purpose of a sensor network. Localization must cost as little as possible while still producing satisfactory results. That means designers must actively work to minimize the power cost, hardware cost, and deployment cost of their localization algorithms.

### 1.3.2 Node density

Many localization algorithms are sensitive to node density. For instance, hop count based schemes generally require high node density so that the hop count approximation for distance is accurate (section 1.2.3). Similarly, algorithms that depend on beacon nodes fail when the beacon density is not high enough in a particular region. Thus, when designing or analyzing an algorithm, it is important to notice the algorithm's implicit density assumptions, since high node density can sometimes be expensive if not totally infeasible.

### 1.3.3 Non-convex topologies

Localization algorithms often have trouble positioning nodes near the edges of a sensor field. This artifact generally occurs because fewer range measurements are available for border nodes, and those few measurements are all taken from the same side of the node. In short, border nodes are a problem because less information is available about them and that information is of lower quality. This problem is exacerbated when a sensor network has a non-convex shape: Sensors outside the main convex body of the network can often prove unlocalizable. Even when locations can be found, the results tend to feature disproportionate error.

### 1.3.4 Environmental obstacles and terrain irregularities

Environmental obstacles and terrain irregularities can also wreak havoc on localization. Large rocks can occlude line of sight, preventing TDoA ranging, or interfere with radios, introducing error into RSSI ranges and producing incorrect hop count ranges. Deployment on grass vs. sand vs. pavement can affect radios and acoustic ranging systems. Indoors, natural features like walls can impede measurements as well. All of these issues are likely to come up in real deployments, so localization systems should be able to cope.

### 1.3.5 System organization

This section defines a taxonomy for localization algorithms based on their computational organization.

Centralized algorithms (section 1.4) are designed to run on a central machine with plenty of computational power. Sensor nodes gather environmental data and pass it back to a base station for analysis, after which the computed positions are transported back into the network. Centralized algorithms circumvent the problem of nodes' computational limitations by accepting the communication cost of moving data back to the base station. This tradeoff becomes less palatable as the network grows larger, however, since it unduly stresses nodes near the base station. Furthermore, it requires that an intelligent base station be deployed with the nodes, which may not always be possible. This scaling problem can be partially alleviated by deploying multiple base stations (forming a multi-tier network).

In contrast, distributed algorithms are designed to run in the network, using massive parallelism and inter-node communication to compensate for the lack of centralized computing power. Often distributed algorithms use a subset of the data to solve for each position independently yielding an approximation of a corresponding centralized algorithm where all the data is considered and used to solve for all the positions simultaneously.

There are two important approaches to distributed localization. The first group, beacon-based distributed algorithms (section 1.5), typically starts with some group of beacons (section 1.2.1). Nodes in the network obtain a distance measurement to a few beacons, then use these measurements to determine their own location. In some algorithms, these newly localized nodes become beacons to help other nodes localize.

The second group approaches localization by trying to optimize a global metric over the network in a distributed fashion. This group splits out into two substantially different approaches. The first approach, relaxation-based distributed algorithms (section 1.6) is to use a coarse algorithm to roughly localize nodes in the network. This coarse algorithm is followed by a refinement step, which typically involves each node adjusting its position to optimize a local error metric. By doing so, these algorithms hope to approximate the optimal solution to a network-wide metric that is the sum of the local error metric at each of the nodes.

Coordinate system stitching (section 1.7) is the second approach to optimizing a network-wide metric in a distributed manner. In these algorithms, the network is divided into small overlapping subregions, each of which creates an optimal local map. Finally, the subregions use a peer-to-peer process to merge their local maps into a single global map. In theory, this global map approximates the global optimum map.

The next four sections treat each of these groups in turn.

## 1.4 CENTRALIZED ALGORITHMS

This section is devoted to centralized localization algorithms. Centralization allows an algorithm to undertake much more complex mathematics than is possible in a distributed setting. However, as we said in the previous section, centralization requires the migration of inter-node ranging and connectivity data to a sufficiently powerful central base station and then the migration of resulting locations back to respective nodes. The main difference between centralized algorithms is the type of processing they do at the base station. We will discuss two types of processing: semidefinite programming and multidimensional scaling.

### 1.4.1 Semidefinite Programming (SDP)

The semidefinite programming (SDP) approach to localization was pioneered by Doherty et al [7]. In this algorithm, geometric constraints between nodes are represented as linear matrix inequalities (LMIs). Once all the constraints in the network are expressed in this form, the LMIs can be combined to form a single semidefinite program. This is solved to produce a bounding region for each node, which Doherty et al simplify to be a bounding box. See figure 1.4 for some sample LMI constraints.

Unfortunately, not all geometric constraints can be expressed as LMIs. In general, only constraints that form convex regions are amenable to representation as an LMI. Thus, angle of arrival data can be represented as a triangle and hop count data can be represented as a circle, but precise range data cannot be conveniently represented, since rings cannot be expressed as convex constraints. This inability to accommodate precise range data may prove to be a significant drawback.

Solving the linear or semidefinite program must be done centrally. The relevant operation is $O(k^2)$ for angle of arrival data, and $O(k^3)$ when radial (e.g. hop count) data is included, where $k$ is the number of convex constraints needed to describe the network. Thus running time is something of an Achilles' heel for this algorithm. A hierarchical version of this algorithm might have better scaling properties, but no relevant performance data has been published to our knowledge.

The real advantage of this algorithm is its elegance. Given a set of convex constraints on a node's position, SDP simply finds the intersection of the constraints. However, SDP's poor scaling and inability to effectively use range data will likely preclude the algorithm's use in practice.

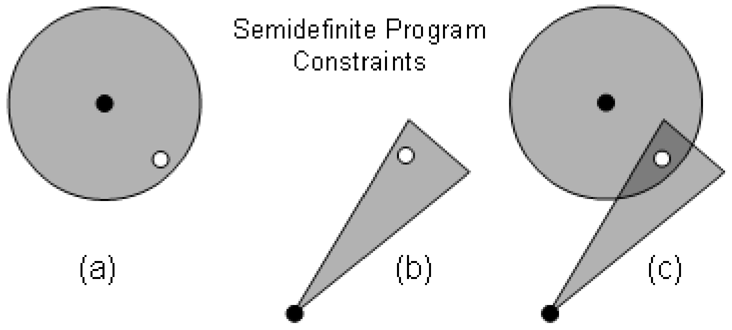

**Fig. 1.4** (a) A radial constraint, for example from radio connectivity. (b) A triangular constraint, for example from angle of arrival data. (c) Location estimate derived from intersection of two convex constraints.

### 1.4.2 MDS-MAP

MDS-MAP is a centralized algorithm due to Shang et al [29]. Instead of using semidefinite programming, however, MDS-MAP uses a technique from mathematical psychology called multidimensional scaling (MDS).

The intuition behind multidimensional scaling is simple. Suppose there are $n$ points, suspended in a volume. We don't know the positions of the points, but we do know the distance between each pair of points. Multidimensional scaling is an $O(n^3)$ algorithm that uses the Law of Cosines and linear algebra to reconstruct the relative positions of the points based on the pairwise distances. The mathematical details of MDS are in appendix C of this chapter.

MDS-MAP is almost a direct application of the simplest kind of multidimensional scaling: classical metric MDS. The algorithm has four stages, which are as follows:

*Step 1* Gather ranging data from the network, and form a sparse matrix $R$, where $r_{ij}$ is the range between nodes $i$ and $j$, or zero if no range was collected (for instance if $i$ and $j$ are physically too far apart).

*Step 2* Run a standard all pairs shortest path algorithm (Dijkstra's, Floyd's) on $R$ to produce a complete matrix of inter-node distances $D$.

*Step 3* Run classical metric MDS on $D$ to find estimated node positions $X$, as described in appendix C.

*Step 4* Transform the solution $X$ into global coordinates using some number of fixed anchor nodes using a coordinate system registration routineB.

MDS-MAP performs well on RSSI data alone, getting performance on the order of half the radio range when the neighborhood size $n_{local}$ is higher than 12. As expected, MDS-MAP estimates improve as ranging improves. MDS-MAP also does not use anchor nodes very well, since it effectively ignores their data until stage 4. As a result, its performance lags behind other algorithms as anchor density increases. The main problem with MDS-MAP, however, is its poor asymptotic performance, which is $O(n^3)$ on account of stages 2 and 3. It turns out that this problem can be partially ameliorated using coordinate system stitching: see section 1.7 for details.

## 1.5 BEACON-BASED DISTRIBUTED ALGORITHMS

In this section we talk about beacon-based distributed algorithms. These algorithms all extrapolate unknown node positions from beacon positions. Thus, they localize nodes directly into the global coordinate space of the beacons. These algorithms are also all distributed, so that all the relevant computation is done on the sensor nodes themselves. We will present four beacon-based distributed algorithms: diffusion, bounding box, gradient multilateration, and APIT.

### 1.5.1 Diffusion

Diffusion arises from a very simple idea: the most likely position of a node is at the centroid of its neighbors positions. Diffusion algorithms require only radio connectivity data. We describe two different variants below.

Bulusu et al [4] localize unknown nodes by simply averaging the positions of all beacons with whom the node has radio connectivity. Thus, Bulusu et al assume that nodes have no way of ranging to beacons. This method is attractive in its blinding simplicity; however, the resulting positions are not very accurate, particularly when beacon density is low, or nodes fall outside the convex hull of their audible beacons.

Fitzpatrick and Meetens [8] describe a more sophisticated variant: each node is at the centroid of its neighbors, including non-beacons. The algorithm is as follows:

*Step 1* Initialize the position of all non-beacon nodes to $(0, 0)$.

*Step 2* Repeat the following until positions converge:

*Step 2a* Set the position of each non-beacon node to the average of all its neighbors' positions.

This variant requires fewer beacon's than Bulusu et al's algorithm; nevertheless, its accuracy is poor when node density is low, nodes are outside the convex hull of the beacons, or node density varies across the network. In all of these cases, a more sophisticated algorithm would improve accuracy dramatically. Fitzpatrick and Meetens' variant also uses substantially more computation than Bulusu et al's approach, since positions must be exchanged between adjacent nodes during step 2.

However, this algorithm is quite useful in networks where nodes are capable of very little computation, but the network topology can be selectively changed to improve localization. In particular, Savvides et al [27] recommend placing some beacons around the edges of the sensor network field. Selectively adding additional beacons can also help resolve pathologies in the diffusion estimates. Bulusu et al [4] describe an approach for adaptive beacon placement to improve diffusion-based localization.

### 1.5.2 Bounding Box

The bounding box algorithm [28, 30] is a computationally simple method of localizing nodes given their ranges to several beacons. See figure 1.5 for an example. Essentially, each node assumes that it lies within the intersection of its beacons' bounding boxes. The bounding box for a beacon $b$ is centered at the beacon position $(x_b, y_b)$, and has height and width $2d_b$, where $d_b$ is the node's distance measurement to the beacon.

The intersection of the bounding boxes can be computed without use of floating point operations:

$$[max(x_i - d_i), max(y_i - d_i)] \times [min(x_i + d_i), min(y_i + d_i)] \tag{1.3}$$

$$i = 1 \ldots n$$

The position of a node is then the center of this final bounding box, as shown in Figure 1.5.

Whitehouse [32] analyzes a distributed version of this algorithm[30], showing that unfortunately this version is highly susceptible to noisy range estimates, especially small estimates which tend to propagate.

The accuracy of the bounding box approach is best when nodes' actual positions are closer to the center of their beacons. Simic and Sastry [30] prove results about convergence, errors, and complexity.

In any event, bounding box works best when sensor nodes have extreme computational limitations, since other algorithms may simply be infeasible. Otherwise, more mathematically rigorous approaches such as gradient multilateration (section 1.5.3) may be more appropriate.

### 1.5.3 Gradient

The principal mathematical operation of the gradient method is called multilateration. Multilateration is a great deal like triangulation, except that multilateration can incorporate ranges from more than three reference points. Formally, given $m$ beacons with known Cartesian positions $b_i$, $i = 1 \ldots m$

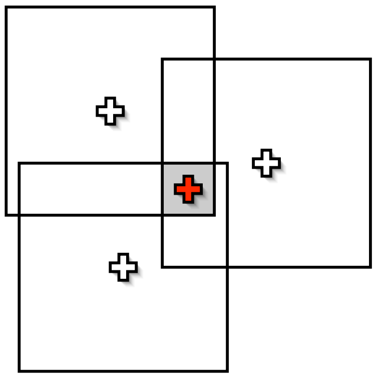

**Fig. 1.5** An example of the intersection of bounding boxes. The center of the intersection is the position estimate for the unknown node. The size of the boxes is based on hop count radio range from the beacons to the unknown node.

and possibly noisy range measurements $r_i$ from the known nodes to an unknown sensor node $s$, multilateration finds the most likely position of $s$. The mathematics of multilateration are outlined in appendix 1.9.

Using gradients to compute ranges for multilateration has been proposed by a number of researchers [5, 23, 4, 16, 1]. These algorithms all assume that there are at least three beacon nodes somewhere in the network (though probably more). Each of these beacon nodes propagates a gradient through the network, which is the distributed equivalent of computing the shortest path distance between all the beacons and all of the unlocalized nodes. The gradient propagation is as follows:

*Step 1* For each node $j$ and beacon $k$, let $d_{jk}$ (the distance from $j$ to $k$) be 0 if $j = k$ and $\infty$ otherwise.

*Step 2* On each node $j$, perform the following steps repeatedly:

*Step 2a* For each beacon $k$ and neighbor $i$, retrieve $d_{ik}$ from $i$.

*Step 2b* For each beacon $k$ and neighbor $i$, apply the following update formula:

$$d_{jk} = \min(d_{ik} + \hat{r}_{ij}, d_{jk})$$

where $\hat{r}_{ij}$ is the estimated distance between nodes $i$ and $j$. These internode distance estimates can be either unweighted (one if there is connectivity, zero otherwise) or measured distances (e.g. using RSSI or TDoA).

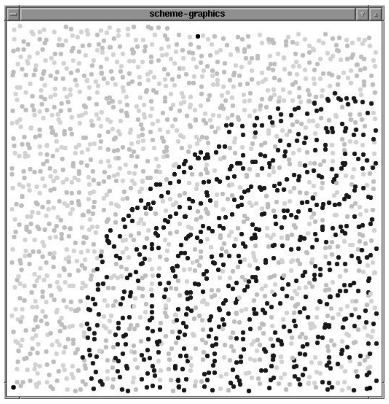

**Fig. 1.6** Gradients propagating from a beacon (in the lower right corner). Each dot represents a sensor node. Sensors are colored based on their gradient value.

After some amount of settling time, each value $d_{jk}$ will be the length of the shortest path between node $j$ and beacon $k$. Figure 1.6 shows the results of running the gradient propagation algorithm with one beacon.

The gradient based distance estimate to a beacon must be adjusted since even given perfect internode distance estimates, gradient distance estimates will always be longer than (or exactly equal) to corresponding straight line distances. Of course given imperfect internode distance estimates, gradient based distance estimate can actually be shorter than straight distances. In fact, Whitehouse [32] shows that it is actually more likely that they are shorter since underestimated internode distances skew all subsequent gradient based estimates. Niculescu and Nath [21] suggest using a correction factor calculated by comparing the actual distance between beacons to the shortest path distances computed during gradient propagation. Each unlocalized node simply applies the correction factor from its closest beacon to its gradient distance estimate.

As an alternative, Nagpal et al [20] in their Amorphous algorithm suggest correcting this distance based on the neighborhood size $n_{local}$, as we previously discussed in section 1.2.3.

Once final distance estimates to beacons have been computed, the actual localization process simply uses multilateration directly on the beacon positions $k$ and the distance measurements $d_{jk}$.

Like the other beacon-based distributed algorithms, this algorithm has the virtue of being direct and easy to understand. It is also scales well (provided the density of beacons is kept constant, otherwise the communication cost can be prohibitive). It is also quite effective in homogeneous topologies where there are few environmental obstructions. However, even when using high quality range data, this algorithm is subject to the deficiencies described in section 1.2.3 and demonstrated in figure 1.2, so it behaves badly in obstructed settings. It also requires substantial node density before its accuracy reaches an acceptable level.

A number of variations to the multilateration approach have been suggested. Niculescu et al [22] suggest propagating AoA information along links. Nagpal et al [19] propose refining the hop count estimates by averaging values among neighbors. This turns out to greatly increase the accuracy of gradient multilateration.

### 1.5.4 APIT

APIT [9] is quite a bit different from the beacon-based distributed algorithms described so far. APIT uses a novel area-based approach, in which nodes are assumed to be able to hear a fairly large number of beacons. However, APIT does not assume that nodes can range to these beacons. Instead, a node forms some number of "beacon triangles", where a beacon triangle is the triangle formed by three arbitrary beacons. The node then decides whether it is inside or outside a given triangle by comparing signal strength measurements with its nearby non-beacon neighbors. Once this process is complete, the node simply finds the intersection of the beacon triangles that contains it. The node chooses the centroid of this intersection region as its position estimate. Figure 1.7 shows an example of this process: each of the triangles represents a triple of beacons and the intersection of all the triangles defines the position of the unknown node.

The actual algorithm is as follows:

*Step 1* Receive beacon positions from hearable beacons.

*Step 2* Initialize *inside-set* to be empty.

*Step 3* For each triangle $T_i$ in possible triangles formed over beacons, add $T_i$ to *inside-set* if node is inside $T_i$. Goto *Step 4* when accuracy of *inside-set* is sufficient.

*Step 4* Compute position estimate as the center of mass of the intersection of all triangles in *inside-set*.

The point in triangle (PIT) test is based on geometry. For a given triangle with points $A$, $B$, and $C$, a given point $M$ is outside triangle $ABC$, if there exists a direction such that a point adjacent to $M$ is further/closer to points

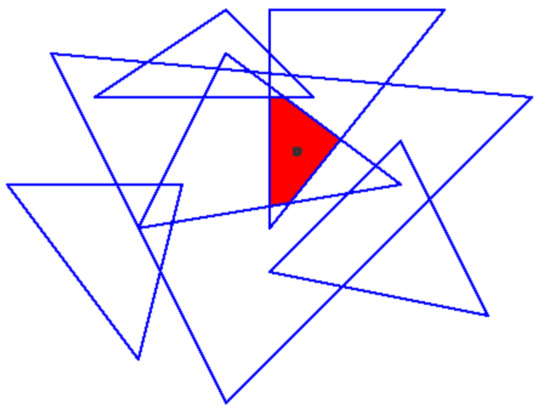

**Fig. 1.7** Node position estimated as the center of mass of the intersection of a number of beacon triangles for which a given node is inside.

$A$, $B$, and $C$ simultaneously. Otherwise, $M$ is inside triangle $ABC$. Unfortunately, given that typically nodes can not move, an approximate PIT (APIT) test is proposed which assumes sufficient node density for approximating node movement. If no neighbor of $M$ is further from/closer to all three anchors $A$, $B$, and $C$ simultaneously, $M$ assumes that it is inside triangle $ABC$. Otherwise, $M$ assumes it resides outside this triangle.

This algorithm is described as being range-free, which means that RSSI range measurements are required to be monotonic and calibrated to be comparable but are not required to produce distance estimates. It could be that the effort put into RSSI calibration would produce an effective enough ranging estimate to be useful for gradient techniques described in section 1.5.3, making the range-free distinction potentially moot. The APIT algorithm also requires a relatively high ratio of beacons to nodes, requires longer range beacons, and is susceptible to erroneously low RSSI readings. On the other hand, He et al [9] show that the algorithm requires smaller amounts of computation and less communication than other beacon based algorithms. In short, APIT is a novel approach which is a potentially promising direction that requires further study.

## 1.6 RELAXATION-BASED DISTRIBUTED ALGORITHMS

This class of algorithms starts with nodes estimating their positions with any of a variety of methods such as gradient distance propagation. These initial positions are then refined from position estimates of neighbors.

Savarese et al [26] refine the initial gradient derived positions using local neighborhood multilateration. Each node adjusts its position by using its neighbors as temporary beacons. Convex optimization can also be used to find an improved position for situations where beacon distance estimates are unavailable.

An equivalent formulation to local multilateration is presented in [24] and is generally referred to as a spring model. This description considers edges between nodes as springs with resting lengths being the actual measured distances. The algorithm involves iteratively adjusting nodes in the direction of their local spring forces. The optimization stops when all nodes have zero forces acting on them. If the magnitude of all the forces between nodes is also zero then the final positions form a global minimum.

Unfortunately, these relaxation techniques are quite sensitive to initial starting positions. Bad starting positions will result in local minima. Priyantha et al [24] describe a technique for producing starting positions for nodes that nearly always avoid bad local minima. The insight is that the network gets tangled and that using the spring model style optimization is unable to fully untangle the network. Their approach starts the network in a "fold-free" state.

The fold-free algorithm works by choosing five reference nodes, one in the center $n_0$ and four on the periphery, $n_1, n_2, n_3, n_4$. The four on the periphery are chosen so that the two pairs $n_1, n_2$ and $n_3, n_4$ are roughly perpendicular to each other. The choice of these nodes is performed using a hop count approximation to distance. The node positions $(x_i, y_i)$ are calculated using polar coordinates $(\theta_i, \rho_i)$ as follows:

$$\begin{aligned}
\theta_i &= h_{0,i}R \\
\rho_i &= \arctan \frac{h_{1,i} - h_{2,i}}{h_{3,i} - h_{4,i}} \\
x_i &= h_{0,i}R\frac{h_{3,i} - h_{4,i}}{l_i} \\
y_i &= h_{0,i}R\frac{h_{1,i} - h_{2,i}}{l_i} \\
l_i &= \sqrt{(h_{3,i} - h_{4,i})^2 + (h_{1,i} - h_{2,i})^2}
\end{aligned} \tag{1.4}$$

where $h_{j,i}$ is the hop count to reference node $j$ and $R$ is the maximum radio range.

These relaxation algorithms have the virtue that they are fully distributed and concurrent and operate without beacons. While the computations are modest and local, it is unclear how well these algorithms scale to much larger networks. Furthermore, there are no provable means for avoiding local minima and local minima problems could worsen at larger scales. To date researchers have avoided local minima by starting optimizations at favorable starting positions, but another alternative would be to utilize optimization techniques, such as simulated annealing [13], which tend to fall into fewer local minima.

## 1.7 COORDINATE SYSTEM STITCHING

In section 1.6, we showed one method of fusing the precision of centralized schemes with the computational advantages of distributed schemes. Coordinate system stitching is a different way of approaching the same problem. It has received a great deal of recent work [6, 21, 17, 18]. Coordinate system stitching works as follows:

*Step 1* Split the network into small overlapping subregions. Very often each subregion is simply a single node and its one-hop neighbors.

*Step 2* For each subregion, compute a "local map", which is essentially an embedding of the nodes in the subregion into a relative coordinate system.

*Step 3* Finally, merge the subregions using a coordinate system registration procedure. Coordinate system registration finds a rigid transformation that maps points in one coordinate system to a different coordinate system. Thus, step three places all the subregions into a single global coordinate system. Many algorithms do this step sub-optimally, since there is a closed-form, fast, and least squares optimal method of registering coordinate systems. We describe this optimal method in section B.

Steps 1 and 2 tend to be unique to each algorithm, whereas Step 3 tends to be the same in every algorithm. We will describe three different methods of performing step 1 and 2, and finally explain the typical method of performing step 3.

Meertens and Fitzpatrick [17] form subregions using one hop neighbors. Local maps are then computed by choosing three nodes to define a relative coordinate system and using multilateration (section 1.5.3) to iteratively add additional nodes to the map, forming a "multilateration subtree".

Moore et al [18] outline an approach which they claim produces more robust local maps. Rather than use three arbitrary nodes to define a map, Moore et al use "robust quadrilaterals" (robust quads), where a robust quad is a fully-connected set of four nodes, where each subtriangle is also "robust". A robust subtriangle must have the property that:

$$b \sin^2 \theta > d_{min}$$

where $b$ is the length of the shortest side, $\theta$ is the size of the smallest angle, and $d_{min}$ is a predetermined constant based on average measurement error. The idea is that the points of a robust quad can be placed correctly with respect to each other (i.e. without "flips"). Moore et al demonstrate that the probability of a robust quadrilateral experiencing internal flips given zero mean Gaussian measurement error can be bounded by setting $d_{min}$ appropriately. In effect, $d_{min}$ filters out quads that have too much positional ambiguity to be localized with confidence. The appropriate level of filtering is based on the amount of uncertainty $\sigma^2$ in the distance measurements.

Once an initial robust quad is chosen, any node that connects to three of the four points in the initial quad can be added using simple multilateration (section 1.5.3). This preserves the probabilistic guarantees provided by the initial robust quad, since the new node forms a new robust quad with the points from the original. By induction, any number of nodes can be added to the local map, as long as each node has a range to three members of the map.

These local maps (which Moore et al call "clusters") are now ready to be stitched together. Optionally, an optimization pass such as those in section 1.6 can be used to refine the local maps first.

Ji et al [12] use multidimensional scaling (MDS) to form local maps. We discussed multidimensional scaling with MDS-MAP in section 1.4.2, and cover the mathematics of MDS in appendix C. Ji et al use an iterative variant of MDS to compensate for missing inter-node distances. This iterative variant turns out to be intimately related to standard iterative least squares algorithms, though it is somewhat more sophisticated. Ji et al focus on RSSI for range data. Once again, subregions are defined to be one-hop neighborhoods.

The stitching phase (step 3 above), uses coordinate system registration (described in section B) in a peer-to-peer fashion to shift all the local maps into a single coordinate system. One way of performing this stitching is described below:

*Step 1* Let the node responsible for each local map choose an integer coordinate system ID at random.

*Step 2* Each node communicates with its neighbors; each pair performs the following steps:

*Step 2a* If both have the same ID, then do nothing further.

*Step 2b* If they have different IDs, then register the map of the node with the lower ID with the map of the node with the higher ID. Afterward, both nodes keep the higher ID as their own.

*Step 3* Repeat step 2 until all nodes have the same ID; now all nodes have a coordinate assignment in a global coordinate system.

Limited work has been done on the mathematical properties of this scheme. Moore et al prove the probability their algorithm constructing correct local maps and prove error lower bounds on the local map positions. Meertens and Fitzpatrick [17] devote some discussion to the topic of error propagation caused by local map stitching. They point out that registering local maps iteratively can lead to error propagation and perhaps unacceptable error rates as networks grow. Furthermore, they argue that in the traditional communication model, where nodes can communicate only with neighbors, this algorithm may converge quite slowly since a single coordinate system must propagate from its source across the entire network. Future work is needed to curb this error propagation.

Furthermore, these techniques have a tendency to orphan nodes, either because they could not be added to a local map or because their local map failed to overlap sufficiently with neighboring maps. Moore et al argue that this is acceptable because the orphaned nodes are the nodes most likely to display high error. However, this answer may not be satisfactory for some applications, many of which cannot use unlocalized nodes for sensing, routing, target tracking, or other tasks.

Nonetheless, coordinate system stitching techniques are quite compelling. They are inherently distributed, since subregion and local map formation can trivially occur in the network and stitching is easily formulated as a peer-to-peer algorithm. Furthermore, they enable the use of sophisticated local map algorithms which are too computationally expensive to use at the global level. For example, map formation using robust quadrilaterals is $O(n^4)$, where $n$ is the number of nodes in the subregion; however, in networks with fixed neighborhood size $n_{local}$, map formation is $O(1)$. Likewise, coordinate system stitching enables the realistic use of $O(n^3)$ multidimensional scaling in sensor networks.

## 1.8 FUTURE DIRECTIONS

The sensor network field and localization in particular are in their infancy. Much work remains in order to address the varied localization requirements of sensor network services and applications. Many future directions stand out as important areas to pursue in order to meet both current and future needs.

Localization hardware will always involve fallible and imperfect components; thus, calibration is imperative [32]. For example, raw measurements from RSSI vary wildly from node to node while most algorithms expect measurements to be at minimum monotonic and comparable. If calibration can bridge this gap, a wide variety of algorithms would become practical on cheap hardware.

Even with accurate calibration, localization hardware produces noisy measurements due to occlusion, collisions, and multipath effects. This mandates an improvement in measurement outlier rejection algorithms. Early work has suggested [15] that outlier rejection can greatly improve the performance of localization algorithms. Some early ideas [15] involve using consistency checks such as symmetry and geometric constraints to reject improbable measurements as discussed in Section 1.2.4. Other possibilities involve using statistical error models to identify outliers.

Future sensor networks will involve movable sensor nodes. New localization algorithms will need to be developed to accommodate these moving nodes. Some algorithms can tolerate a certain amount of movement but more experiments and algorithm development is required. Some researchers [31, 4] have touched on this issue with adaptive beacon placement, but much more work is needed.

No current localization algorithm adequately scales for ultra-scale sensor networks (i.e., 10000 nodes and beyond). It seems likely that such networks will end up being multi-tiered, and will require the development of more hierarchical algorithms.

## 1.9 CONCLUSION

In this chapter we presented the foundations of sensor network localization. We discussed localization hardware, issues in localization algorithm design, major localization techniques, and future directions. In this section, we summarize the tradeoffs and provide guidelines for choosing different algorithms based on context and available hardware.

The first primary distinction between algorithms is those that require beacons (described in section 1.5) and those that do not (described in sections 1.4, 1.6, and 1.7). Beaconless algorithms necessarily produce relative coordinate systems which can optionally be registered to a global coordinate system by positioning three (or four) nodes. Often sensor network deployments make the use of beacons prohibitive and furthermore many applications do not require a global coordinate system. In these situations beaconless algorithms suffice. Finally, some algorithms (such as APIT from section 1.5.4) require a higher beacon to node ratio than others to achieve a given level of accuracy.

The next distinction between localization algorithms is their hardware requirements. All sensor nodes have radios and most can measure signal strength, thus, algorithms that rely on hop count or RSSI require the least hardware. Varying degrees of ranging precision can be achieved from RSSI, with hop count being at the low end, with one bit precision. Gradient algorithms (from section 1.5.3 such as DV-hop and Amorphous can often produce quite accurate results using only hop counts and sufficient node density. Sometimes, a microphone and speaker are required for other reasons, making the use of more accurate TDoA ranging possible. Sometimes nodes lack sufficient arithmetic processing making certain algorithms impractical. Algorithms such as bounding-box and APIT make the least demands on processors (although APIT makes some demands on memory).

Finally, certain algorithms are centralized while others are distributed. Centralized algorithms typically compute more exact positions and can be

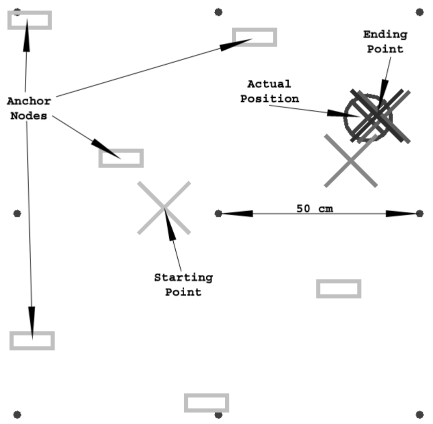

**Fig. A.1** In this diagram, a single unknown node with ranges to six different beacons localizes itself using multilateration. The ground truth position of the unknown node is circled. The X's mark the best estimate after each iteration of least squares, with darker colors indicating higher iterations.

competitive in situations where accuracy is important and the exfiltration of ranging data and dissemination of resulting location data is not prohibitively time consuming nor error prone. Centralized algorithms could actually be a viable option in many typical deployments where a base station is already needed for other reasons. Distributed algorithms are often local approximations to centralized algorithms, but have the virtue that they do not depend on a large centralized computer and potentially have better scalability.

Other issues to consider are battery life and communication costs. Often these two are intertwined as typically communication is the most battery draining sensor node activity. Consult He et al [9] for a comparison of communication costs (and other metrics) of a number of localization algorithms.

The development of localization algorithms is proceeding at a fast pace. While the task appears simple, to compute positions for each node in a sensor network, the best algorithm depends heavily on a variety of factors such as application needs and available localization hardware. Future algorithms will address new sensor network needs such as mobile nodes and ultra-scale sizes.

## Appendix: A. Multilateration

This appendix derives a solution to the multilateration problem (section 1.5.3). See figure A.1 to see an example of this solution in practice.

Multilateration is a simple technique, but the specific mathematics of its implementation vary widely, as do its application in sensor networks. The purpose of multilateration is simple: given $m$ nodes with known Cartesian positions $b_i$, $i = 1 \ldots m$ and possibly noisy range measurements $r_i$ from the known nodes to an unknown node $s$, multilateration finds the most likely position of $s$.

Multilateration is typically done by minimizing the squared error between the observed ranges $r_i$ and the predicted distance $\|s - b_i\|$:

$$\begin{aligned} s &= \operatorname*{argmin}_s E(s) \\ E(s) &= \sum_{i=1}^{m} (\|s - b_i\| - r_i)^2 \end{aligned} \tag{A.1}$$

This minimization problem can be solved using Newton-Raphson/least squares as follows. First, approximate the error function $e(s, b_i) = \|s - b_i\| - r_i$ in equation (A.1) with a first order Taylor series about $s_0$:

$$\begin{aligned} e(s, b_i) &\approx e(s_0, b_i) + \nabla e(s_0, b_i)(s - s_0) \\ &= \nabla e(s_0, b_i)s - (-e(s_0, b_i) + \nabla e(s_0, b_i)s_0) \\ \nabla e(s, b_i) &= \frac{s - b_i}{\|s - b_i\|} \end{aligned}$$

Plug this approximation back into equation (A.1):

$$s \approx \operatorname*{argmin}_s \sum_{i=1}^{m} (\nabla e(s_0, b_i)s - (-e(s_0, b_i) + \nabla e(s_0, b_i)s_0))^2$$

Stacking terms:

$$s \approx \operatorname*{argmin}_s \|As - b\|^2 \tag{A.2a}$$

$$A = \begin{bmatrix} \nabla e(s_0, b_1) \\ \nabla e(s_0, b_2) \\ \vdots \\ \nabla e(s_0, b_m) \end{bmatrix} \tag{A.2b}$$

$$b = \begin{bmatrix} -e(s_0, b_1) + \nabla e(s_0, b_1)s_0 \\ -e(s_0, b_2) + \nabla e(s_0, b_2)s_0 \\ \vdots \\ -e(s_0, b_m) + \nabla e(s_0, b_m)s_0 \end{bmatrix} \tag{A.2c}$$

The right side of equation (A.2a) is in exactly the right form to be solved by an off-the-shelf iterative least squares solver. The resulting $s$ is a good estimate of the unknown sensor's position, provided $b_i$ and $r_i$ are accurate. Here is a summary of the multilateration method:

*Step 1* Choose $s_0$ to be a starting point for the optimization. The choice is somewhat arbitrary, but the centroid $\bar{b}$ is a good one:

$$\bar{b} = \frac{1}{m} \sum_{i=1}^{m} b_i$$

*Step 2* Compute $A$ and $b$ using $s_0$ and equations (A.2b) and (A.2c).

*Step 3* Compute $s_0' = \underset{x}{\operatorname{argmin}} \|Ax - b\|^2$ using a least squares solver.

*Step 4* If $E(s_0) - E(s_0') < \epsilon$, then $s_0'$ is the solution, otherwise set $s_0 = s_0'$ and return to Step 2.

There are many ways to solve the multilateration problem. The one presented above is equivalent to Newton-Raphson descent on the error function $E$ (equation (A.1)). Most alternate methods also attempt to minimize squared error using some form of iterative optimization. To see a prototypical example of an algorithm that uses multilateration, see section 1.5.3.

## Appendix: B. Coordinate System Registration

Many localization algorithms compute a relative coordinate assignment for a group of sensors and later transform this local coordinate assignment into a different coordinate system. To do this, the algorithm must compute a translation vector, a scale factor, and an orthonormal rotation matrix that define the transformation from one coordinate system to the other. The process of finding these quantities is known as “Coordinate System Registration.” Registration can be performed for two dimensions as long as three points have known coordinates in both systems. The three dimensional version naturally requires four points.

We will present Horn et al's method of solving the coordinate system registration problem. It has many advantages over commonly used registration methods:

1. It has provable optimality over the canonical least squares error metric (equation (B.2)).

2. It uses *all* the data available, though it can compute a correct result with as few as three (or four) points.

3. It can be computed quickly, since its running time is proportional to the number of common points $n$.

There is one caveat: even after a rigid transformation, it is unlikely that the known points will precisely align, since the measurements used to localize the points are likely to have errors. Thus, the best that can be done is a minimization of the misalignment between the two coordinate systems. Let $x_{l,i}$ and $x_{r,i}$ be the known positions of node $i = 1 \ldots n$ in the left hand and right hand coordinate systems respectively. The goal of registration is to find a translation $t$, scale $s$, and rotation $R$ that transform a point $x$ in the left hand coordinate system to the equivalent point $x'$ in the right hand coordinate system using the formula:

$$x' = sRx + t \tag{B.1}$$

Horn et al approach this problem using squared error; they look for a $t$, $s$, and $R$ that meet the following condition:

$$(t, s, R) = \underset{t,s,R}{\operatorname{argmin}} \sum_{i=1}^{n} \|e_i\|^2 \tag{B.2a}$$

$$e_i = x_{r,i} - sRx_{l,i} - t \tag{B.2b}$$

In [11], Horn et al derive a closed form for equation (B.2) which can be computed in $O(n)$ time. The method is outlined below with emphasis on the precise steps required to perform the computation. For more detail on the mathematical underpinnings, see [11]. To see the method in action, see figure B.1.

*Step 1* Compute the centroids of $x_l$ and $x_r$:

$$\bar{x}_l = \frac{1}{n}\sum_{i=1}^{n} x_{l,i} \qquad \bar{x}_r = \frac{1}{n}\sum_{i=1}^{n} x_{r,i}$$

*Step 2* Shift the points so that they are defined with respect to the centroids:

$$x'_{l,i} = x_{l,i} - \bar{x}_l \qquad x'_{r,i} = x_{r,i} - \bar{x}_r$$

Now the error term in equation (B.2b) can be rewritten as:

$$e_i = x'_{r,i} - sRx'_{l,i} - t'$$
$$t' = t - \bar{x}_r + sR\bar{x}_l$$

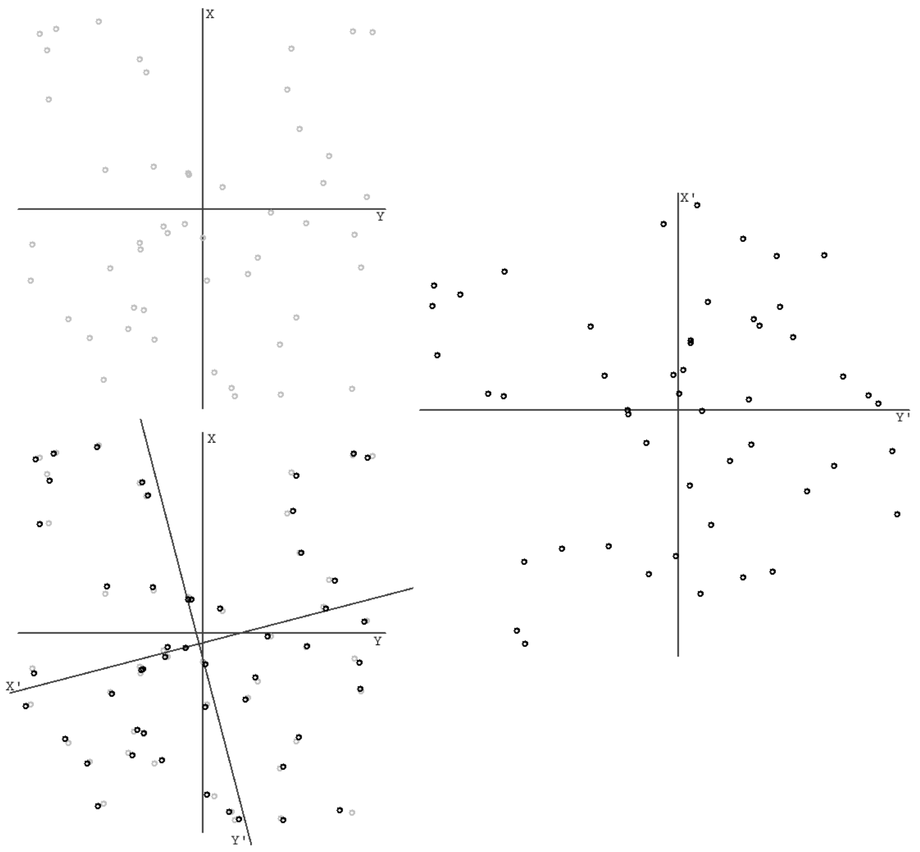

**Fig. B.1** An example of coordinate system registration. In the upper left is a set of reference points $(X, Y)$. On the right, the reference points have been moved into a new coordinate system by a linear transformation $(X', Y') = L(X, Y)$ and then jittered to simulate position error. Finally, in the lower right the $(X', Y')$ coordinate system is brought into registration with the reference coordinate system $(X, Y)$.

As it turns out, the squared error from equation (B.2) is minimized when $t' = \mathbf{0}$, independent of $s$ and $R$. Therefore:

$$t = \bar{x}_r - sR\bar{x}_l \tag{B.3}$$

So after $s$ and $R$ have been computed, equation (B.3) can be used to compute $t$. Since $t' = \mathbf{0}$, the error term can be rewritten as:

$$e_i = x'_{r,i} - sRx'_{l,i} \tag{B.4}$$

Now that $t$ is out of the way, we can focus on finding $s$ and $R$. Equation (B.4) can be rewritten as:

$$e_i = \frac{1}{\sqrt{s}} x'_{r,i} - \sqrt{s} R x'_{l,i} \tag{B.5}$$

So now we need only find:

$$\begin{aligned} (s, R) &= \underset{s,R}{\operatorname{argmin}} \sum_{i=1}^{n} \|e_i\|^2 \\ &= \underset{s,R}{\operatorname{argmin}} \frac{1}{s} \sum_{i=1}^{n} \|x'_{r,i}\|^2 + s \sum_{i=1}^{n} \|r_{l,i}\|^2 \\ &- 2 \sum_{i=1}^{n} x'_{r,i} \cdot (Rx'_{l,i}) \end{aligned} \tag{B.6}$$

By completing the square in $s$, it can be shown that equation (B.6) (and thus equation (B.2)) is minimized when:

$$s = \sqrt{\sum_{i=1}^{n} \|x'_{r,i}\|^2 / \sum_{i=1}^{n} \|x'_{l,i}\|^2} \tag{B.7}$$

*Step 3* Use equation (B.7) to compute the optimal scale factor $s$. Now equation (B.6) can be simplified to:

$$R = \underset{R}{\operatorname{argmin}}\, 2 \left( \sqrt{\left( \sum_{i=1}^{n} \|x'_{r,i}\|^2 \right) \left( \sum_{i=1}^{n} \|x'_{l,i}\|^2 \right)} - \sum_{i=1}^{n} x'_{r,i} \cdot (Rx'_{l,i}) \right) \tag{B.8}$$

Equation (B.8) is minimized when the following is true:

$$R = \underset{R}{\operatorname{argmax}} \sum_{i=1}^{n} x'_{r,i} \cdot (R x'_{l,i})$$

This is the same as:

$$R = \underset{R}{\operatorname{argmax}} \operatorname{Trace}(R^T M) \tag{B.9a}$$

$$M = \sum_{i=1}^{n} x'_{r,i} (x'_{l,i})^T \tag{B.9b}$$

$M$ is a 2x2 or 3x3 matrix, depending on whether the points $x_{l,i}$ and $x_{r,i}$ are two or three dimensional. For the remainder of this discussion, assume $M$ is 3x3; the results are similar for the two dimensional case.

*Step 4* Compute $M$ using equation (B.9b).

*Step 5* Compute the eigen-decomposition of $M^T M$. That is, find eigenvalues $\lambda_1$, $\lambda_2$, $\lambda_3$ and eigenvectors $\hat{u}_1$, $\hat{u}_2$, $\hat{u}_3$ such that:

$$M^T M = \lambda_1 \hat{u}_1 \hat{u}_1^T + \lambda_2 \hat{u}_2 \hat{u}_2^T + \lambda_3 \hat{u}_3 \hat{u}_3^T$$

*Step 6* Compute $S = (M^T M)^{1/2}$ and $U = M S^{-1}$. That is:

$$S = \sqrt{\lambda_1} \hat{u}_1 \hat{u}_1^T + \sqrt{\lambda_2} \hat{u}_2 \hat{u}_2^T + \sqrt{\lambda_3} \hat{u}_3 \hat{u}_3^T$$

$$U = M S^{-1} = M \left( \frac{1}{\sqrt{\lambda_1}} \hat{u}_1 \hat{u}_1^T + \frac{1}{\sqrt{\lambda_2}} \hat{u}_2 \hat{u}_2^T + \frac{1}{\sqrt{\lambda_3}} \hat{u}_3 \hat{u}_3^T \right)$$

Note that $M = US$, and that $U$ is orthonormal, since $U^T U = I$.

We can now write $\operatorname{Trace}(R^T M)$ from equation (B.9a) as:

$$\begin{aligned} \operatorname{Trace}(R^T U S) = & \sqrt{\lambda_1} \operatorname{Trace}(R^T U \hat{u}_1 \hat{u}_1^T) \\ & + \sqrt{\lambda_2} \operatorname{Trace}(R^T U \hat{u}_2 \hat{u}_2^T) \\ & + \sqrt{\lambda_3} \operatorname{Trace}(R^T U \hat{u}_3 \hat{u}_3^T) \end{aligned}$$

$\operatorname{Trace}(R^T U \hat{u}_i \hat{u}_i^T)$ can be rewritten as $(R\hat{u}_i \cdot U\hat{u}_i)$. Since $\hat{u}_i$ is a unit vector, and since U and R are orthonormal transformations, $(R\hat{u}_i \cdot U\hat{u}_i) \leq 1$, with equality only when $R\hat{u}_i = U\hat{u}_i$. Therefore:

$$\operatorname{Trace}(R^T U S) \leq \sqrt{\lambda_1} + \sqrt{\lambda_2} + \sqrt{\lambda_3} = \operatorname{Trace}(S)$$

The maximum value of Trace($R^TUS$) occurs when $R^TU = I$, i.e. when $R = U$. Therefore, the rotation $R$ necessary to minimize the error in equation (B.8) is given by:

$$R = U = M\left(\frac{1}{\sqrt{\lambda_1}}\hat{u}_1\hat{u}_1^T + \frac{1}{\sqrt{\lambda_2}}\hat{u}_2\hat{u}_2^T + \frac{1}{\sqrt{\lambda_3}}\hat{u}_3\hat{u}_3^T\right) \tag{B.10}$$

*Step 7* Compute $R$ using equation (B.10). $R$ is an orthonormal matrix that encapsulates the rotation and possible reflection necessary to transform $x_{l,i}$ into $x_{r,i}$.

*Step 8* Now we have $R$ and $s$, so use equation (B.3) to compute $t$. $R$, $s$, and $t$ form a complete linear transformation between the two coordinate systems that minimizes equation (B.2).

*Step 9* For each point $x$ in the left hand coordinate system, compute the corresponding position $x'$ in the right hand coordinate system using:

$$x' = t + sRx$$

Even though this math may look imposing, it is straightforward to implement, and gives provably optimal results. As you will see shortly, many algorithms depend on coordinate system registration, either to shift a completely localized relative topology into global coordinates, or to "stitch together" small local topologies into a single consistent coordinate assignment. This appendix described a powerful closed-form method of performing the necessary registration operations.

## Appendix: C. Multidimensional Scaling

Multidimensional Scaling (MDS) was originally developed for use in mathematical psychology. It comes in many variations, but all the variations share a common goal. Given a set of points whose position is unknown and measured distances between each pair of points. Multidimensional scaling determines the underlying dimensionality of the points, and finds an embedding of the points in that space that honors the pairwise distances between them.

Clearly, MDS has potential in the sensor localization domain. Using only ranging data, without anchors or GPS, MDS can solve for the relative coordinates of a group of sensor nodes with resilience to measurement error and rather high accuracy.

This section focuses on a type of multidimensional scaling called "classical metric MDS," classical because it uses only one matrix of "dissimilarity" or distance information, and metric because the dissimilarity information is quantitative (e.g. distance measurements), as opposed to ordinal. There are many other types, but they are not common in sensor networks so they are omitted for brevity.

Let there be $n$ sensors in a network, with positions $X_i$, $i = 1 \dots n$, and let $X = [X_1, X_2, \dots, X_n]^T$. $X$ is $nxm$, where $m$ is the dimensionality of $X$. For now, consider $m$ to be an unknown. Let $D = [d_{ij}]$ be the $nxn$ matrix of pairwise distance measurements, where $d_{ij}$ is the measured distance between $X_i$ and $X_j$ for $i \neq j$, and $d_{ii} = 0$ for all $i$. The distance measurements $d_{ij}$ must obey the triangular inequality: $d_{ij} + d_{ik} \geq d_{jk}$ for all $(i, j, k)$.

The goal of MDS is to find an assignment of $X$ in low-dimensional space that minimizes a "Stress function", defined as:

$$X = \underset{X}{\operatorname{argmin}} \operatorname{Stress}(X) \tag{C.1}$$

$$\operatorname{Stress}(X) = \sqrt{\frac{\sum_{i=1}^{n} \sum_{j=1}^{i-1} (d_{ij} - \delta_{ij})^2}{\sum_{i=1}^{n} \sum_{j=1}^{i-1} \delta_{ij}^2}} \tag{C.2}$$

In equation (C.1), $\delta_{ij}$ is the distance between $X_i$ and $X_j$. Thus, the metric MDS stress function is closely related to the squared error function we have seen in other techniques such as multilateration (section 1.5.3).

Classical metric multidimensional scaling is derived from the Law of Cosines, which states that given two sides of a triangle $d_{ij}$, $d_{ik}$, and the angle between them $\theta_{jik}$, the third side can be computed using the formula:

$$d_{jk}^2 = d_{ij}^2 + d_{ik}^2 - 2 d_{ij} d_{ik} \cos \theta_{jik} \tag{C.3}$$

Rewriting:

$$d_{ij} d_{ik} \cos \theta_{jik} = \frac{1}{2}(d_{ij}^2 + d_{ik}^2 - d_{jk}^2) \tag{C.4}$$

The left side of equation (C.4) can be rewritten as a dot product:

$$(X_j - X_i) \cdot (X_k - X_i) = \frac{1}{2}(d_{ij}^2 + d_{ik}^2 - d_{jk}^2) \tag{C.5}$$

If all measurements are perfect, then a good zero-stress way to solve for the positions $X$ is to choose some $X_0$ from $X$ to be the origin of a coordinate system, and construct a matrix $B_{(n-1)x(n-1)}$ as follows:

$$b_{ij} = \frac{1}{2}(d_{0i}^2 + d_{0j}^2 - d_{ij}^2) \tag{C.6}$$

$B$ is known as the matrix of scalar products. As we know from equation (C.5), we can write $B$ in terms of $X$. Call $X'_{(n-1)xm}$ the matrix $X$ where each of the $X_i$'s is shifted to have its origin at $X_0$: $X'_i = X_i - X_0$. Then, using equations (C.5) and (C.6):

$$X'X'^T = B$$

We can solve for $X'$ by taking an eigen-decomposition of $B$ into an orthonormal matrix of eigenvectors and a diagonal matrix of matching eigenvalues:

$$\begin{aligned} B &= X'X'^T = UVU^T \\ X' &= UV^{1/2} \end{aligned} \tag{C.7}$$

The problem is that $X'$ has too many columns: we need to find $X$ in 2-space or 3-space. To do this, we throw away all but the two or three largest eigenvalues from $V$, leaving a 2x2 or 3x3 diagonal matrix, and throw away the matching eigenvectors (columns) of $U$, leaving $U_{(n-1)x2}$ or $U_{(n-1)x3}$. Then $X'$ has the proper dimensionality.

Note that this method produces a coordinate system that is a linear transformation from the coordinate system of the true $X_i$'s. Reconciling the two requires a registration procedure like that of appendix B.

Remember, though, that we said this method only works when the data $d_{ij}$ is perfect, which is an unrealistic assumption. In practice, there is some error, which ends up in the stress value of the final coordinate assignment. Fortunately, the classical metric MDS method generalizes to gracefully cover measurement errors. Above, we chose a single point from our data to be the origin. This choice gives $X_0$ an undue influence on the error of $X$. Thus, real MDS doesn't use a point from the data; rather, it uses a special point in the center of the $X_i$'s. This point is found by "double centering" the squared distance matrix. The squared distance matrix $D^2 = [d_{ij}^2]$. To double center a matrix, subtract the row and column means from each element. Then, add the grand mean to each element. Finally, multiply by $-1/2$. The element-wise formula for double centering is below.

$$\begin{aligned} b_{ij} &= -\frac{1}{2}\left(d_{ij}^2 - \frac{1}{n}\sum_{k=1}^{n} d_{kj}^2 - \frac{1}{n}\sum_{k=1}^{n} d_{ik}^2 + \frac{1}{n^2}\sum_{k=1}^{n}\sum_{l=1}^{n} d_{kl}^2\right) \\ &= \sum_{a=1}^{m} x_{ia}x_{ja} \end{aligned} \tag{C.8}$$

Reformulating equation (C.8) in matrix notation:

$$B_{nxn} = -\frac{1}{2} J D^2 J = X X^T \tag{C.9a}$$

$$J_{nxn} = I_{nxn} - \frac{1}{n} e^T e \tag{C.9b}$$

$$e_{1xn} = [1, 1, 1, \ldots, 1] \tag{C.9c}$$

Equation (C.9) is an expression for $X$ in terms of $D$, in $m$-dimensional space. If $m = n - 1$, then there is a trivial assignment of $X_1 \ldots X_n$ that makes Stress($X$) $= 0$. As $m$ decreases, it turns out that Stress($X$) must increase or stay the same; it cannot decrease. We know that the measurements $D$ originate from a two or three dimensional space. If the measurements from $D$ are perfect, then there is a zero stress assignment of $X$ when $m = 2$ or 3. However, measurement error makes it unlikely that such an assignment really exists. Thus, some stress is inevitable as we reduce the dimensionality from $n$ to 2 or 3.

As before, this dimensionality reduction is done by taking an eigen-decomposition of $B$, then removing eigenvalues and eigenvectors. This is a safe operation because $B$ is symmetric positive definite, and therefore has $n$ positive eigenvalues.

$$\begin{aligned} B = XX^T &= UVU^T \\ X &= UV^{1/2} \end{aligned} \tag{C.10}$$

Thus, multidimensional scaling provides a method of converting a complete matrix of distance measurements to a matching topology in 2-space or 3-space. This conversion is quite resilient to measurement error, since increased measurement error simply becomes an increase in the stress function. To see an example of MDS in action, look at figure C.1.

Unfortunately, multidimensional scaling has some disadvantages. First, the main computation of MDS, the eigen-decomposition of $B$ (equation (C.10)) requires $O(n^3)$ time. As a result, a single pass of multidimensional scaling cannot operate on a large topology, particularly in the constrained computational environment of sensor networks. Second, classical MDS requires that $D$ contain a distance measurement for *all* pairs of nodes. This requirement is impossible to meet with ranging hardware alone in large networks; thus, implementations of MDS in sensor networks must do pre-processing on measured data to generate $D$ (section 1.4.2) or use coordinate system stitching to distribute the computation (1.7).

To conclude, here are the steps of classical metric multidimensional scaling:

*Step 1* Create the symmetric matrix $D = [d_{ij}]$, with $d_{ii} = 0$ and $d_{ij} + d_{ik} \geq d_{jk}$.

*Step 2* Create the symmetric matrix $J$ (equation (C.9b)).

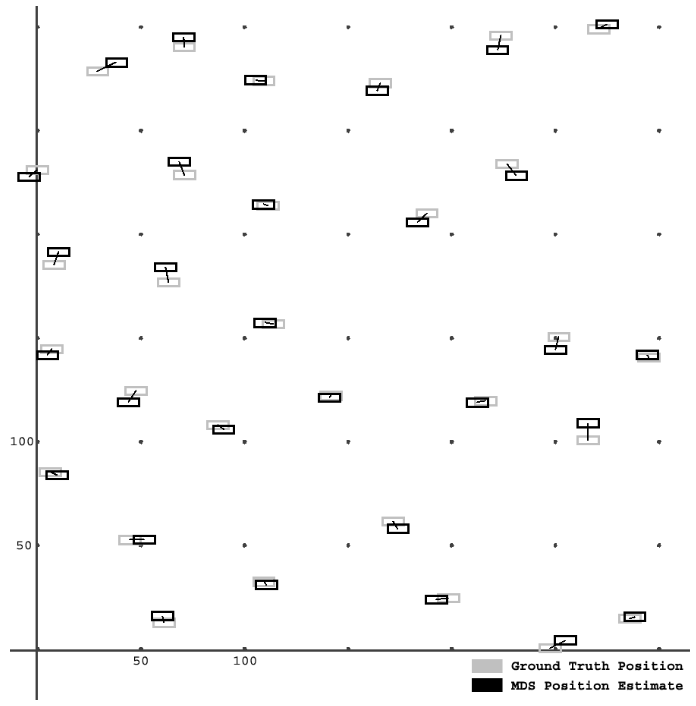

**Fig. C.1** Topology constructed by multidimensional scaling. Each inter-node range measurement has zero-mean Gaussian error with a standard deviation of 10 units.

*Step 3* Compute $B$ using $D^2 = [d_{ij}^2]$ and $J$ (equation (C.9a)).

*Step 4* Take an eigen-decomposition $UVU^T$ of $B$.

*Step 5* Let $V_d$ be the diagonal matrix of the $d$ largest eigenvalues in $V$, where $d$ is the desired dimensionality of the solution.

*Step 6* Let $U_d$ be the $d$ eigenvectors from $U$ that match the eigenvalues in $V_d$.

*Step 6* Compute $X_d = [X_1, X_2, \ldots, X_n]^T$ using $X_d = U_d V_d^{1/2}$. $V_d^{1/2}$ can be computed by taking the square root of each of $V_d$'s diagonal elements.

*Step 7* (Optional) Transform the $X_i$'s from $X_d$ into the desired global coordinate space using some coordinate system registration algorithm (appendix B). These transformed $X_i$'s are the solution.

## REFERENCES

1. *Organizing a Global Coordinate System from Local Information on an Ad Hoc Sensor Network*, April 2003.
2. P. Bahl and V. Padmanabhan. Radar: An in-building rf-based user location and tracking system. In *INFOCOMM*, 2000.
3. H. Balakrishnan, R. Baliga, D. Curtis, M. Goraczko, A. Miu, N. Priyantha, A. Smith, K. Steele, S. Teller, and K. Wang. Lessons from developing and deploying the cricket indoor location system. Preprint., November 2003.
4. Nirupama Bulusu, Vladimir Bychkovskiy, Deborah Estrin, and John Heidemann. Scalable, ad hoc deployable rf-based localization. In *Grace Hopper Celebration of Women in Computing Conference 2002, Vancouver, British Columbia, Canada.*, October 2002.
5. William Joseph Butera. *Programming a paintable computer.* PhD thesis, mit, 2002.
6. Srdan Capkun, Maher Hamdi, and Jean-Pierre Hubaux. GPS-free positioning in mobile ad-hoc networks. In *HICSS*, 2001.
7. L. Doherty, L. El Ghaoui, and K. S. J. Pister. Convex position estimation in wireless sensor networks. In *Proceedings of Infocom 2001*, April 2001.
8. Stephen Fitzpatrick and Lambert Meertens. Diffusion based localization. private communication, 2004.
9. T. He, C. Huang, B. Blum, J. Stankovic, and T. Abdelzaher. Range-free localization schemes in large scale sensor networks, 2003.

10. J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister. System architecture directions for networked sensors. In *Proceedings of ASPLOS-IX*, 2000.

11. B.K.P. Horn, H. Hilden, and S. Negahdaripour. Closed-form solution of absolute orientation using orthonormal matrices. *Journal of the Optical Society of America A*, 5(7), 1988.

12. X. Ji and H. Zha. Sensor positioning in wireless ad hoc networks using multidimensional scaling. In *Infocom*, 2004.

13. S. Kirkpatrick, C. D. Gelatt Jr., and M. P. Vecchi. Optimization by simulated annealing, 1983.

14. L. Kleinrock and J.A. Silvester. Optimum transmission radii for packet radio networks or why six is a magic number. In *IEEE National Telecommunications Conference*, December 1978.

15. Y. Kwon, K. Mechitov, S. Sundresh, W. Kim, and G. Agha. Resilient localization for sensor networks in outdoor environments. Technical Report UIUCDCS-R-2004-2449, University of Illinois at Urbana-Champaign, June 2004.

16. James D. McLurkin. Algorithms for distributed sensor networks. Master's thesis, UCB, December 1999.

17. Lambert Meertens and Stephen Fitzpatrick. The distributed construction of a global coordinate system in a network of static computational nodes from inter-node distances, 2004.

18. David Moore, John Leonard, Daniela Rus, and Seth Teller. Robust distributed network localization with noisy range measurements. In *Proceedings of ACM Sensys-04*, Nov 2004.

19. R. Nagpal. Organizing a global coordinate system from local information on an amorphous computer, 1999.

20. R. Nagpal, H. Shrobe, and J. Bachrach. Organizing a global coordinate system from local information on an ad hoc sensor network. In *IPSN*, 2003.

21. D. Niculescu and B. Nath. Ad hoc positioning system (APS), 2001.

22. D. Niculescu and B. Nath. Ad hoc positioning system (APS) using AOA, 2003.

23. D. Niculescu and B. Nath. Localized positioning in ad hoc networks, 2003.

24. N. Priyantha, H. Balakrishnan, E. Demaine, and S. Teller. Anchor-free distributed localization in sensor networks, 2003.

25. N. Priyantha, A. Miu, H. Balakrishnan, and S. Teller. The cricket compass for context-aware mobile applications. In *MOBICOM*. ACM, July 2001.

26. C. Savarese, J. Rabaey, and J. Beutel. Locationing in distributed ad-hoc wireless sensor networks, 2001.

27. A. Savvides, H. Park, and M. Srivastava. The bits and flops of the n-hop multilateration primitive for node localization problems, 2002.

28. Andreas Savvides, Chih-Chieh Han, and Mani B. Strivastava. Dynamic fine-grained localization in ad-hoc networks of sensors. In *Mobile Computing and Networking*, pages 166–179, 2001.

29. Shang, Ruml, Zhang, and Fromherz. Localization from mere connectivity. In *MobiHoc*, 2003.

30. S. Simic and S. Sastry. Distributed localization in wireless ad hoc networks, 2002.

31. Seth Teller and Eric Demaine etc. Mobile beacon placement etc, 2004.

32. Cameron Whitehouse. The design of calamari: an ad-hoc localization system for sensor networks. Master's thesis, University of California at Berkeley, 2002.

# Index