Distributed Fine-Grained Node Localization in Ad-Hoc Networks

Andreas Savvides and Mani B. Srivastava
Networked and Embedded Systems Lab
Electrical Engineering Department
University of California, Los Angeles
{asavvide,mbs}@ee.ucla.edu

September 12, 2002

Abstract

The recent advances in MEMS, embedded systems and wireless communication technologies are making the realization and deployment of networked wireless microsensors a tangible task. Vital to the success of wireless microsensor networks is the ability of microsensors to "collectively perform sensing and computation". In this paper, we study one challenge in microsensor networks that requires collective sensing and computation, node localization. The node localization algorithms presented here, enable ad-hoc deployed sensor nodes to accurately estimate their locations by using known beacon locations that are several hops away and distance measurements to neighboring nodes. To prevent error accumulation in the network, node locations are computed by setting up and solving a global non-linear optimization problem. The solution is presented in two computation models, centralized and a fully distributed approximation of the centralized model. Our simulation results show that using the fully distributed model, resource constrained sensor nodes can collectively solve a large non-linear optimization problem that none of the nodes can solve individually without compromising localization accuracy. This approach results in significant savings in computation and communication, that allows fine-grained localization to run on a low cost sensor node we have developed.

Keywords: Ad-Hoc Localization, Distributed Localization, Sensor Networks

1 Introduction

Driven by the recent advances in electronics and communication technologies, the cross pollination of many different disciplines is causing an explosion in the application space of wirelessly connected devices. In a few years time, it is expected that the number of wireless devices per person will be in the hundreds and possibly thousands. These devices will operate autonomously and will move about their environments to perform different tasks in

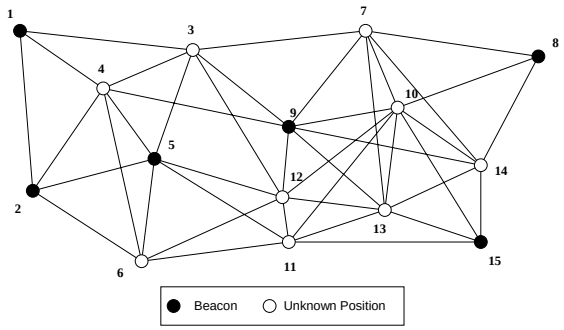


Figure 1: Typical multihop network setup

the background without human intervention. Such tasks will include sensing, computation, communication and actuation.

In many situations where large numbers of wireless devices equipped with sensors and actuators are scouted for a particular mission, the physical locations of nodes in space are inherently coupled to the effectiveness of the entire network. In such context, knowledge of location is needed to accurately report the origins of events, to help operate and manage the network through geographic-aware addressing and routing [17, 32, 16], and can help with network coverage assessment [18]. We envision that, when endowed with the appropriate localization mechanisms, tiny wireless sensors will make the fundamental building blocks of many smart environments such as the Smart Kindergarten described in [27].

Motivated by the above requirements, we seek to develop a set of ad-hoc localization techniques with which wireless devices can accurately determine their locations in space. These techniques are primarily targeted for indoor environments and outdoor environments where the use of GPS receivers is prohibitive either because of insufficient line-of-sight with GPS satellites or merely because the use of GPS receivers is rendered infeasible due to size and power constraints of wireless devices [4].

In this paper we describe a localization algorithm that enables sensor-equipped nodes

that are found multiple hops away from other nodes with known locations (the *beacons*) to accurately estimate their physical locations by collaborating with each other. Figure 1 shows a typical network setup. Nodes connected by a line are within radio range of each other and they can also measure distances to each other. The dark nodes are aware of their locations and act as beacons for the remaining nodes. First, the nodes with unknown locations use their onboard sensors to measure distances to their neighbors. Later on, nodes exchange information on their measurements and known beacon locations. Using this information, nodes compute their locations with high accuracy by setting up and solving a global non-linear optimization problem. The use of a global vantage point takes all known measurement and location information constraints into account and ensures that error accumulation in the network is minimized. The solution is presented in two computation models, centralized and distributed that can accommodate different network setups: globally centralized, locally centralized and fully distributed. Furthermore, we show that this type of computation can be done very efficiently inside the network in a fully distributed manner. This enables inexpensive nodes with constrained computing capabilities, to collaboratively solve a global non-linear optimization problem, a task that none of the nodes can perform individually. This fully distributed method is also tolerant to node failures and communication cost is evenly shared across all the nodes.

Our algorithms are validated on a combined ns-2 and MATLAB simulation testbed. The sensor node parameters used in the simulations are derived from a low cost wireless sensor node equipped with ultrasonic ranging we have specifically developed for studying node localization problems. Besides ultrasonic ranging used in our experiments, these algorithms are also applicable to other ranging technologies such as ultra-wide band [7] and RF time-of-flight [2].

The remainder of this paper is organized as follows. Next section contains a preliminary

discussion on the ad-hoc localization challenges and introduces our hardware platforms. Section 3 overviews our solution components. Section 4 describes the initial setup configuration for collaborative multilateration. Section 5 explains centralized and distributed position refinement. Section 6 shows our simulation results and section 7 surveys the related work. Section 8 concludes the paper.

2 Preliminaries

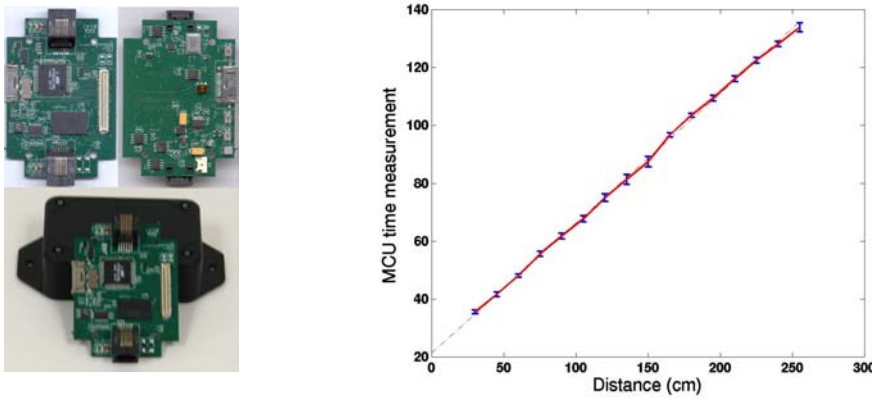
In the setup we investigate, the node localization problem is stated as follows.

Given a network of sensor nodes where only a small fraction of the nodes is aware of their initial locations (the beacons), estimate the locations of the remaining nodes (the unknowns).

For localization purposes, we assume that each node is equipped with a ranging system, which can measure distances between two nodes, and its range is equal or less than the node's radio range. Furthermore, in the scenarios we study all the nodes are allowed to collaborate with each other without any privacy or security concerns. Before presenting the solution in detail, we first discuss *atomic multilateration* and *iterative multilateration* that are later on used as core primitives for our multihop localization scheme, *collaborative multilateration*.

2.1 An Example Sensor Node

Figure 2a shows Medusa MK-2, an experimental sensor node that we have developed for studying node localization problems. It is equipped with a low power radio from RF Monolithics, an 8-bit ATMega128L microcontroller from Atmel running at 4MHz that runs base band processing for the radio, provides the interface for sensors and performs some lightweight computation. For more demanding computation tasks such as the least squares localization algorithms presented in this paper, the Medusa MK-2 node uses a more powerful



(a) (b)

Figure 2: a) Medusa MK-2 node, b) Ultrasound Ranging Characterization

16/32 bit 40MHz AT91RF4081 ARM THUMB microcontroller also from Atmel. With this configuration, the node can carry a wide variety of sensors including a ranging subsystem for localization purposes.

The ranging subsystem is made up of a set of 40KHz ultrasonic transducers that measure distances between a pair of nodes by recording the time-of-arrival between simultaneously transmitted RF and ultrasound pulses. This ranging system can measure distances with 2-centimeter accuracy. The maximum range is set to three meters to allow for multihop experiments in a lab setting. Figure 2b shows the ranging characterization of our system. The other capabilities of the Medusa MK-2 node are described in detail in [25]

For other applications longer-range, high precision ranging technologies are also available. Hexamite Inc [11] advertises 20 meter range devices based on modulated ultrasound that can achieve up to 0.3 centimeters accuracy. Intersense Soni-Disks [13] use time-of-flight measurements between coordinated infrared and 40KHz ultrasonic transmissions to perform measure distances with 0.5 centimeters accuracy. Also for larger systems, SICK's [26] laser range finders have an effective range of 80 meters and 1 centimeter accuracy.

2.2 Atomic Multilateration

In its simplest form, node localization is similar to the GPS problem [15], where the role of the beacon nodes is similar to that of GPS satellites. In a two dimensional setup, if three (four for three dimensions) or more non-collinear beacon nodes (GPS satellites in the case of GPS) are within range of an unknown node, the unknown node can measure its distance from the beacons and use the known beacon locations to estimate its position. The estimation is done by minimizing the difference between measured and estimated distances using least-squares. In an ad-hoc network, since one cannot guarantee that each unknown node will have at least three beacons as neighbors, a different method for estimating node positions is required.

2.3 Iterative Multilateration

Iterative multilateration is a scheme we have originally proposed in [24]. It derives node locations over multiple hops by allowing nodes with newly estimated locations to act as beacons for other nodes with unknown locations. In figure 1 for example, node 10 can estimate its location using beacons $8, 9$ and 15. After this, node 10 can act as a beacon for node 7 which estimates its location using nodes $8, 9$ and 10 as beacons. By performing an atomic multilateration operation iteratively through the network, more and more nodes can estimate their locations. Although, this solution is initially appealing it suffers from two problems. First, the error associated with atomic multilateration at each step accumulates in the network, resulting in degradation of the estimates. Second, this process can easily get stuck if the beacon density is insufficient to guarantee that each node in the network will have at least three neighbors, which are either beacons or will eventually become beacons as part of the iterative multilateration process. To mitigate this problem, in [24] we proposed the development of another method, collaborative multilateration, that we describe next.

2.4 Collaborative Multilateration

Collaborative multilateration enables nodes that are not directly connected to beacon nodes to collaborate with other intermediate nodes with unknown locations situated between themselves and the beacons to jointly estimate their locations. To prevent error accumulation, collaborative multilateration operates on over-constrained subsets of the network topology.

Collaborative multilateration estimates node locations in three main phases: 1) formation of collaborative subtrees, 2) computation of initial estimates, 3) position refinement. During the first phase, the nodes organize themselves into groups, called the collaborative subtrees. Each collaborative subtree incorporates enough topology constraints to ensure that the position of each unknown node can be uniquely determined. From an optimization perspective, this implies that there are enough constraints so that the node position problem is well-constrained or over-constrained and can provide a unique solution for the location of each unknown node inside the collaborative subtree. By forming a system of at least n nonlinear equations and n unknown variables to be determined, collaborative subtrees ensure that each unknown member of the computation subtree has a unique possible solution. The nodes that do not meet the criteria for collaborative subtrees cannot participate in this configuration. The position estimates for such nodes are determined later in a post-processing phase. In the second phase, each unknown node computes an initial estimate of its location based on the known beacon locations and the inter-node distance measurements conducted by the nodes using their onboard sensors. These initial estimates are used to initialize the refinement process in the third phase. The third phase computes a least squares estimate of the node locations. Finally, a post-processing phase uses the computed node estimates to refine the position estimates of nodes that could not participate in the computation subtree configuration. This phase is actually a repetition of the second phase, but the remaining unknown nodes have more constraints contributed by the newly estimated positions.

The first and second phases constitute the initial configuration for the position refinement process. These phases are independent from each other and can take place in parallel in an actual network. Phase three can start as soon as the first two phases are completed and it can terminate at different stages depending on the demands of the application. If computation is done at a central point (either a central computer for the whole network, or a local cluster head), the process will terminate when the unknown nodes receive their position estimates. If the distributed computation form is used, then the processes termination depends on the localization accuracy demands of the application. If the application requires just an indication of proximity, the localization process can only perform the second phase and terminate. If more accurate localization is required, the distributed computation continues until required precision is achieved. The first two phases are described in detail in the next section and phase 3 is discussed in section 4.

3 INITIAL CONFIGURATION

3.1 Phase 1: Collaborative Subtrees

A computation subtree is a configuration of nodes with unknown locations and beacons for which the solution to the position estimates of the unknowns can be uniquely determined. This is achieved by obtaining a well determined or preferably over-determined set of equations - n variables to be estimated and at least n equations. Collaborative subtrees have another desirable property that facilitates a distributed computation model. This will be described in section 4.2. To determine the requirements for solution uniqueness we develop our discussion by reviewing the requirements of the single hop multilateration. Later on, we augment these requirements to cover the multihop case.

3.1.1 One-Hop Multilateration Requirements

In the single hop setup of figure 3a, the basic requirement for one unknown node to have a unique solution on a 2D plane is that it is within range of at least three beacons. If the beacons lie in a straight line, the node configuration is symmetric, and there is more than one possible solution.

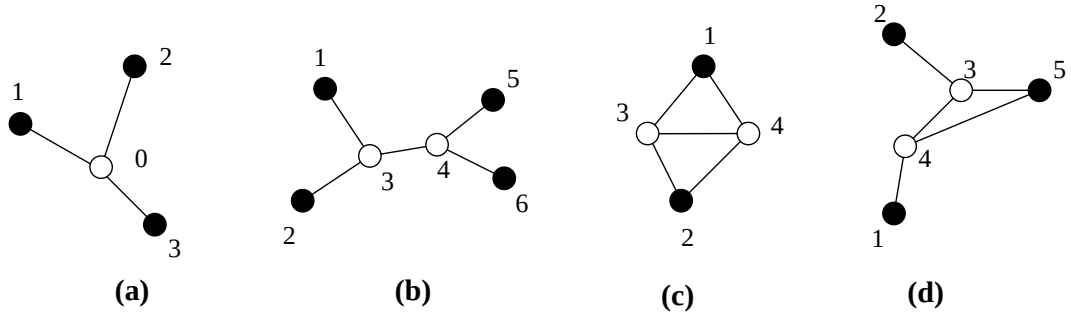


Figure 3: a) One-hop multilateration, b)Two-hop multilateration, c) Symmetric case, d) Each unknown has one independent reference

3.1.2 Two-Hop Multilateration Requirements

Using the one-hop multilateration requirements as a starting point, the corresponding set of requirements for a two-hop multilateration can be established. A two-hop multilateration represents the case where the beacons are not always directly connected to the node but they are within a two-hop radius from the unknown node. In this situation, two or more unknown nodes can utilize the beacon location information and the intermediate distance measurements between themselves and the beacons to jointly estimate their locations. Like the one-hop case, each unknown node needs to be connected to at least three nodes, but these nodes are not required to be beacons. Instead, unknown nodes need to query their neighbors and determine which of their neighbors have only one possible position solution

and use them as reference points to determine if their position solution is unique. To achieve this we introduce the notion of *tentatively unique* nodes. A node is said to be tentatively unique if its position solution is unique based on the assumption that the position of the node that queries it is unique. From this perspective, a node being queried is tentatively unique if in addition to the node that queries it has at least two neighbors that are either beacons or their solutions are tentatively unique. Figure 3b illustrates the most basic case. Nodes 3 and 4 are unknown and they are both connected to three nodes. Note that from the perspective of node 3, one of its links terminates to an unknown, node 4. Node 4 however, has two more outgoing links to beacons 5 and 6. If we assume that node 3 has a unique position solution, then node 4 also has a unique position solution. If however, node 4 has a unique position solution, then the position of node 3 is also unique because it is connected to 3 nodes with unique solutions - 1,2 and 4. This condition is necessary but not sufficient to guarantee that there is only one possible node position estimate. Many symmetric topologies that meet the above requirement can yield more than one possible position estimate.

Condition 1 *To have a unique possible position solution, it is necessary that an unknown node be connected to at least three nodes that are either beacons or have tentatively unique positions.*

The first symmetric case follows from the conditions of the single hop setup - the nodes with tentatively unique solutions used as references for an unknown should not lie in a straight line. If the reference points lie in a straight line, then the unknown node will have two possible positions so the solution to the location estimate is not unique.

Condition 2 *It is necessary for an unknown node to use at least one reference point that is not collinear with the rest of its reference points.*

Although the positions of the reference points are not known, one can test for this condition using basic trigonometry. In figure 4, assuming that nodes A,C and D are known to have unique solutions, node B tries to establish if its position solution is unique. To do so node B computes the angles ABC, CBD and ABD. Using the angle ABD, node B can calculate the distance $\bar{AD}$. If the computed distance AD is equal to the sum of distances AC and CD then the nodes are collinear [1]. In such a case, node B decides that its solution is not unique.

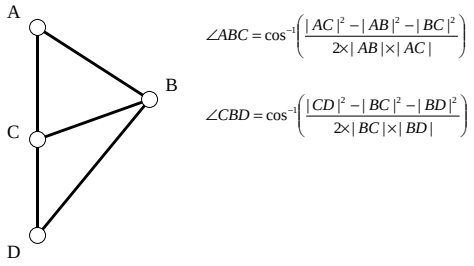


Figure 4: Detecting collinear configurations

Another type of setup that can cause symmetry problems is shown in figure 3c. Nodes 3 and 4 both have 3 links to nodes with tentatively unique positions but the setup is symmetric since the two nodes can be swapped without any violation of the constraints imposed by the intra-node distance measurements. To avoid this situation where the whole network can be rotated over two pivot points (nodes 1 and 2 in figure 3d) an additional condition is set.

Condition 3 *In each pair of unknown nodes that use the link to each other as a constraint, it is necessary that each node has at least one link that connects to a different node from the nodes used as references by the other node.*

[1] Here we loosely use the term 'equal' for clarity and simplicity of the explanation, in practice we also need to consider the noise incurred by the distance measurement process

The network in figure 3d is an example configuration that satisfies condition 3. Both unknown nodes 3 and 4 have at least one independent reference. Node 4 has beacon 1 and node 3 has beacon 2. The above three necessary conditions tend to be sufficient for most of the practical cases we have tried in our simulations, but we have not yet proven this conjecture analytically.

3.1.3 N-Hop Multilateration Requirements

To determine if nodes located within n hops from the beacons have unique solutions a similar set of criteria is applied. Starting from an unknown node we test if it has at least three neighbors with tentatively unique positions. If the node has three neighbors that do not already know if their solution is unique, then a recursive call is executed at each neighbor to determine if its position is unique. To meet the requirements of condition 3 each node used as an independent reference is marked and placed in a used node set U. This prevents other nodes from subsequent recursive calls to re-use that node as an independent reference. At every step, each node checks if the criteria for condition 2 are also met. This recursive algorithm is given in figure 5. The algorithm initially starts at an unknown node u and constructs a computation subtree by traversing the network in a depth first manner checking for the multilateration requirements. At each node the algorithm assumes that the previous node (i.e the *caller*) has a tentatively unique solution and, tries to find out whether the node currently visited has a tentatively unique solution. The algorithm terminates when a computation subtree is formed. At this point any unknown nodes outside the computation subtree whose position can be determined solely by considering the locations of the nodes in the computation subtree are also absorbed into computation tree. This adds more constraints to the refinement process and yield further improvement to the position estimates.

```
Inputs: node id, node id of the caller and a boolean
flag that specifies if the caller node is the node that initiated
the computation tree search.
Output: if success full a list of edges that
make up the computation subtree, otherwise, it returns NULL
------------------------------------------------------------
list isUnique(u, caller, initiator)
begin
if we are at the initiator
  begin U :=0 ; m := 0 end
else begin U := 1; m := 1 end
for each beacon neighbor b begin
    if b is marked begin
      if b is not the only beacon neighbor
      of the node that marked it begin
        mark b as visited by u;
        decrement U for the node that previously
         marked b; U := U + 1
      end
      else begin
        if m < 2 begin
          m := m + 1; U := U + 1;
        end
      end
    end
    else begin
      edge_list ← edge(b, u); U := U + 1
      mark b as visited by u
    end
end
for each unknown marked neighbor z begin
    if m < 2 begin
      edge_list ← edge(u, z)
      m := m + 1 ; U := U + 1
    end
    if U = 3 return edge_list
end
for each unknown unmarked neighbor x
  if (edge_listB := isUnique(x, u, false)) not NULL
    edge_list ← edge(x, u)
    edge_list ← edge_listB
    U := U + 1
    if U = 3 return edge_list
  end
end
return NULL
end
```

Figure 5: Forming collaborative subtrees

3.2 Phase 2: Obtaining Initial Estimates

The initial estimates are obtained by applying the distance measurements as constraints on the x and y coordinates of the unknown nodes. Figure 6 shows how the distance measurement from two beacons A and B can be used to obtain the x coordinate bounds for the unknown node C. In figure 6, if the distance between an unknown and the beacon A is a then the x coordinates of node C are bounded by a to the left and to the right of the x coordinate of beacon A, $x_A - a$ and $x_A + a$. Beacon B imposes its constraints in the same way. Similarly, if beacon B is two hops away from C (as in figure 6, the coordinates bounds of node C are defined by the length of the minimum weight path to C, $b + c$, so the bounds for C's x-coordinates with respect to B are $x_B - (b+c)$ and $x_B + (b+c)$. By knowing this information C can determine that its x coordinate bounds with respect to beacons A and B are $x_B + (b+c)$ and $x_A - a$. This operation selects the tightest left hand side bound from and the tightest right hand side bound from each beacon. The same operation is applied on the y coordinates. The node then combines its bounds on the x and y coordinates, to construct a bounding box of the region where the node lies. To obtain this bounding box, the locations of all the beacons are forwarded to all unknowns along a minimum weight path. This forwarding is similar to distance vector routing (e.g DSDV [20]) but using the measured distances instead of hops as weights.

The initial position estimate of a node is taken to be as the center of the bounding box. When these constraints are combined with the conditions for position uniqueness, they provide a good set of initial estimates for the Kalman Filter used in the next phase. The resulting initial estimates for a 10 node network with 3 beacons is shown in figure 6c. The initial estimates for each node (shown as crosses in figure 6) are close to the actual node positions and provide a good starting point for the refinement process. One challenge in this method is that the quality of the initial estimates suffers when the unknown nodes lie far

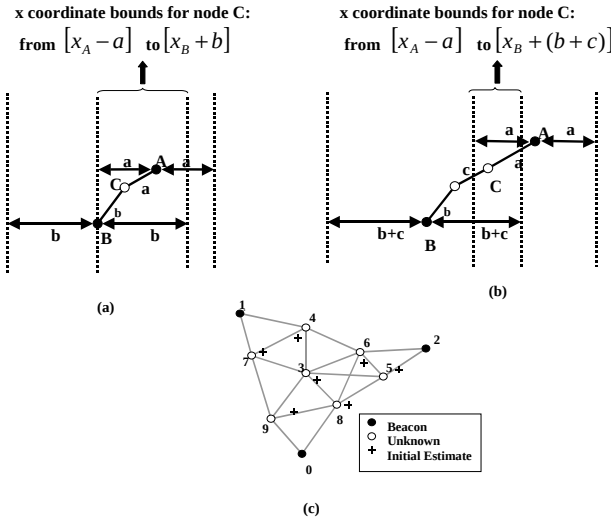


Figure 6: Initial estimates over multiple hops a) One-hop schenario, b) Two-hop schenario, c) Example network

outside the convex hull formed by the beacons on the perimeter of the collaborative subtree, then the initial estimates degrade. In our experiments with large networks we deploy some of the beacons on the edges of the sensor field.

4 Phase 3: Position Refinement

In the third phase, the initial node positions are refined, using least-squares estimation. Our implementation uses a Kalman Filter, which provides the same location estimates as iterative least squares in a static network [9]. The Kalman Filter was chosen because of its ability to fuse measurements from multiple sensing modalities and to track the nodes after the localization process is complete. Position refinement can be implemented in one of two computation models, centralized and distributed that are described next.

4.1 Computing at a Central Node

Using the collaborative subtrees and the initial position estimates, the unknown node position estimates can be computed at a central point. The edges of the computation subtree give

a well-determined or over-determined set of equations, which can be solved using non-linear optimization. The non-linear set of equations for the network in figure 3b is shown in equations 1 [2]. As in the one hop case, the objective is to minimize the residuals between the measured distances between the nodes and the distances computed using the node location estimates.

$$\begin{aligned} f_{2,3} &= R_{2,3} - \sqrt{(x_2 - ex_3)^2 + (y_2 - ey_3)^2} \\ f_{3,5} &= R_{3,5} - \sqrt{(ex_3 - x_5)^2 + (ey_3 - y_5)^2} \\ f_{4,3} &= R_{4,3} - \sqrt{(ex_4 - ex_3)^2 + (ey_4 - ey_3)^2} \\ f_{4,5} &= R_{4,5} - \sqrt{(ex_4 - x_5)^2 + (ey_4 + y_5)^2} \\ f_{4,1} &= R_{4,1} - \sqrt{(ex_4 - x_1)^2 + (ey_4 - y_1)^2} \end{aligned} \tag{1}$$

The $R_{i,j}$ quantities represent the measured distances between two nodes and the quantities under the square root indicate the estimated distances. $f_{i,j}$ represent the residual between the measured and estimated quantities. The objective function in equation 2 is to minimize the mean square error over all equations. This differs from its one-ho counterpart in the sense that distance measurements between unknown nodes are also used as constraints.

$$F(x_3, y_3, x_4, y_4) = min \sum f_{i,j}^2 \tag{2}$$

The solution to this optimization problem can be obtained using some of the standard least squares methods. In our implementation, this is done using a Kalman Filter.

4.1.1 Kalman Filter Implementation

A Kalman Filter consists of two phases, a time update phase and a measurement update phase. The former is a prediction in time (equations 3, and 4). This prediction of node positions in time presented by $\hat{x}_k^-$ is based on a known model of the system behavior, represented

[2] the prefix e in front of x, y denotes estimated coordinates, as opposed to known coordinates

by matrix A. u_k is a zero mean gaussian random variable and B is the error covariance matrix for this random variable. P_k^- is the apriori estimate of the error covariance and Q is the process noise. The latter is an update of the current time estimate based on a measurement that was just obtained. K represents the Kalman Filter gain and it serves a weight to the residual of the filter. The residual is the difference between the measurement, (represented by z_k) and the predicted measurement $H\hat{x}_k^-$. $\hat{z}_k$ the distance between nodes, based on the current position estimate. Matrix H is the Jacobian of $\hat{z}_k$ with respect to the apriori estimates (found in $\hat{x}_k^-$) of the locations. Matrix R is the measurement noise covariance matrix. This contains the known noise covariance of the distance measurement system (i.e based on the characterization of our ultrasonic system we assume white gaussian noise with standard deviation = 20 mm). $\hat{x}_k$ is the new estimate obtained after the prediction and measurement are combined. This new prediction measurement has a new error covariance matrix P. Matrix I stands for the identity matrix. A more detailed discussion of Kalman Filters and be found in [30, 3].

For the purposes of collaborative multilateration, the network is assumed to be static. Since the positions of the nodes, do not change in time, the time update phase is not used. Based on this, the discussion in this section focuses on the second part of the Kalman Filter, the measurement update phase.

$$\hat{x}_k^- = A\hat{x}_{k-1} + Bu_k \tag{3}$$

$$P_k{}^- = AP_{k-1}A^T + Q \tag{4}$$

$$K_k = P_k{}^- H^T (H P_k{}^- H^T + R)^{-1} \tag{5}$$

$$\hat{x}_k = K_k(z_k - H\hat{x}_k)^{-1} \tag{6}$$

$$P_k = (I - K_k H)P_k^- \tag{7}$$

To estimate the unknown locations, our algorithm proceeds as follows:

1. Set the vector $\hat{x}_k^-$ to the initial estimates obtained in section 3.2
2. Evaluate equations 5, 6 and 7 - the measurement update phase
3. Evaluate the stopping criterion $\sqrt{(\hat{x}_k)^2 - \left(\hat{x}_k^-\right)^2} \leq \Delta$ where Δ is some predefined tolerance (0.01 in our experiments). If the criterion is met then the algorithm terminates and has the new position estimates. Otherwise,
4. Set the prediction $\hat{x}_k^-$ to the new estimate $\hat{x}_k$ and restart from step 2.

The term Δ is used as an indicator of the gradient of the Kalman Filter and shows how far the process is from convergence. For illustrative purposes we provide the setup of the Kalman Filter in terms of figure 3c. The initial estimates (denoted by ex and ey) are placed in vector, $\hat{x}_k^-$, so $\hat{x}_k^- = [ex_3, ey_3, ex_4, ey_4]$. The ranging measurements are placed in vector z_k, $z_k = [R_{2,3}, R_{3,5}, R_{3,4}, R_{4,1}, R_{4,5}]$. Vector $\hat{z}_k$ contains the ranging distances based on the current estimates

$$\hat{z}_k^T = \begin{bmatrix} \sqrt{(x_2 - ex_3)^2 + (y_2 - ey_3)^2} \\ \sqrt{(ex_3 - x_5)^2 + (ey_3 - y_5)^2} \\ \sqrt{(ex_3 - ex_4)^2 + (ey_3 - ey_4)^2} \\ \sqrt{(ex_4 - x_1)^2 + (ey_4 - y_1)^2} \\ \sqrt{(ex_4 - x_5)^2 + (ey_4 - y_5)^2} \end{bmatrix}$$

Matrix H is the jacobian of $\hat{z}_k$ with respect to $\hat{x}_k^-$.

$$H = \begin{bmatrix} \frac{ex_3 - x_2}{\hat{z}(1)} & \frac{ey_3 - y_2}{\hat{z}_k(1)} & 0 & 0 \\ \frac{ex_3 - x_5}{\hat{z}_k(2)} & \frac{ey_3 - ey_5}{\hat{z}_k(2)} & 0 & 0 \\ \frac{ex_3 - ex_4}{\hat{z}_k(3)} & \frac{ey_3 - ey_4}{\hat{z}_k(3)} & \frac{ex_4 - ex_3}{\hat{z}_k(3)} & \frac{ey_4 - ey_3}{\hat{(z)}_k(3)} \\ \frac{ex_4 - x_1}{\hat{z}_k(4)} & \frac{ey_4 - y_1}{\hat{z}_k(4)} & 0 & 0 \\ \frac{ex_4 - x_5}{\hat{z}_k(5)} & \frac{ey_4 - y_5}{\hat{z}_k(5)} & 0 & 0 \end{bmatrix}$$

The above illustration shows how the matrix size and subsequently the amount of computation increases with the number of nodes. Each edge in the collaborative subtree contributes one entry in the measurement matrix z_k. In matrix H, the number of unknown nodes determines the number of columns and the number of edges determines the number of rows. Each beacon-unknown edge adds two entries to the row and each unknown-unknown edge adds four entries. The noise covariance matrices P_k^- and P_k are square matrices whose size depends on the number of unknown nodes and the measurement noise matrix R is a square matrix and it size determined by the number of edges in the collaborative subtree. These changes in matrix sizes dramatically increase the amount of computation that has to be performed. Furthermore, from our simulation experience, we noted that when the Kalman Filter has more variables to estimate, it takes more iterations to converge. Unfortunately, since the estimation process is an iterative process we cannot quantify the amount of computation required for the filter to converge analytically. Instead, we evaluate this empirically in our simulations by measuring the number of FLOPS MATLAB consumes [3] until the Kalman Filter converges. This description also shows that although node positions can be estimated accurately using this method, such computation cannot be performed using a low cost microcontroller available on the sensor nodes. To facilitate this type of computation, we have

[3]Note that the FLOPS measure obtained from MATLAB is used as a means for relative comparison of the computation cost between our centralized and distributed schemes. In our actual implementation, the Kalman Filters are implemented in C and they execute on the node processor.

developed a distributed approximation where every node participates in the computation.

4.2 Computing at Every Node

In the distributed version, of our algorithm, computation is spatially distributed across the network and each unknown node is responsible for computing its own location estimate. This is achieved by performing local computation and communication with the neighboring nodes. This idea is similar to using a distributed Kalman Filter [31, 22], but also has the following differences:

- The Kalman Filter executes in the context of a computation subtree and each node executes a one hop multilateration based on its distance measurements and the location information from its neighbors.
- Instead of using a decentralized Kalman Filter, an approximation in which nodes do not exchange covariance information is used. This conserves energy since it reduces communication.
- The computation is applied over a multihop network thus it needs to be integrated in the ad-hoc networking protocols.

The underlying principle of our distributed scheme is that after the completion of the first two phases of the localization process, each node inside the collaborative subtree computes an estimate of its location. Since most unknown nodes, are not directly connected to beacons, they use the initial location estimates (obtained in section 3.2) of their neighbors as the reference points for estimating their locations. This is similar to applying collaborative multilateration in the context of a collaborative subtree. As soon as an unknown computes a new estimate, it broadcasts this estimate to its neighbors, and the neighbors use it to update their own position estimates. This computation is repeated from node to node across the

network until all the nodes reach the pre-specified tolerance Δ, $\left(\sqrt{(\hat{x}_k)^2 - \left(\hat{x}_k^-\right)^2} \leq \Delta\right)$. Figure 7 is a pictorial representation of the computation process. First node 4 computes its location estimate using beacons 1 and 5 and node 3 as references. Once node 4 broadcasts its update, node 3 recomputes its own estimate using beacons 2 and 5 and the new estimate received from node 4. Node 3 then broadcasts the new estimate and node 4 uses this to compute a new estimate that is more accurate than its previous estimate.

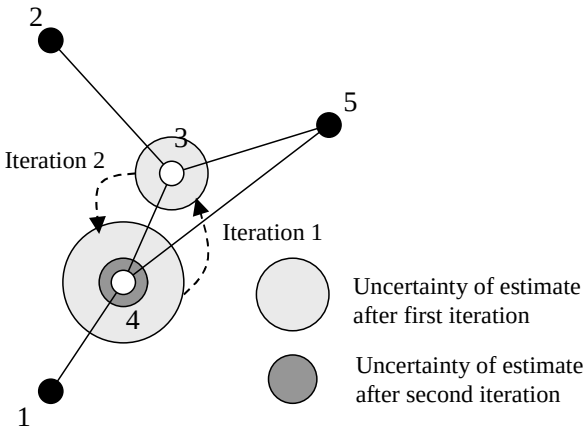


Figure 7: Initial estimates over multiple hops

4.2.1 Driving Distributed Collaborative Multilateration

If the collaborative subtree contains more than two nodes then the order in which multilaterations execute at each node needs to be controlled. If this process proceeds uncontrolled, then the nodes will converge at local minimal and erroneous estimates will be produced. Imagine a computation subtree with many unknown nodes (i.e 20). If two neighboring unknown nodes A and B that compute and broadcast their updates as soon as an update from each other is received, then their updating process will proceed faster than the remaining nodes in the computation subtree. This introduces a "local oscillation" in the computation that makes the nodes converge to their final estimates much faster but without complying with the global gradient, thus yielding erroneous estimates.

To prevent this problem, the multilaterations at each node are executed in a sequence across all the unknown members of the computation subtree. This sequence is repeated until the multilaterations of all the members of the computation subtree converge to a pre-specified tolerance. The in-sequence execution of the multilaterations inside the computation subtree establishes a gradient with respect to the global topology constraints at each node, thus enabling the node to compute its global optimum locally.

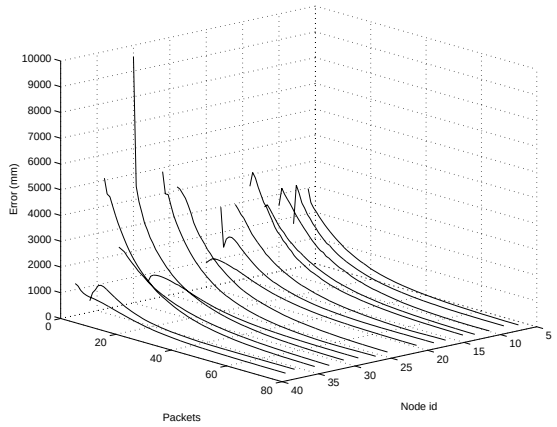


Figure 8: Progress of distributed computation

Figure 8 is an excerpt from our combined ns-2-matlab simulation. As we will describe in the next section, we developed our simulations in ns-2. To be able to measure FLOPS, we implemented the Kalman Filters in MATLAB and then using the MATLAB compiler and the MATLAB C++ libraries, we translated the filters to C++ so we can use them at the routing layer in ns-2. The figure demonstrates the execution of distributed position refinement on a network of 34 nodes and 6 beacons [4]. The unknown nodes in the network have an average degree of 4, 15 meters range and the 20 mm white gaussian noise in the distance measurement system. The x-axis shows the number of packets (estimate updates) transmitted by each node, the node ids are shown in the y-axis and the z-axis shows the error in millimeters. The values of the error before any packets are transmitted, at packets = 0 on

[4] For clarity and good visibility purposes the graph only shows how the process proceeds on the even numbered nodes.

the x-axis, represent the errors of the initial estimates (obtained in section 3.2). As it can be seen form the figure, each node starts at different levels of error. In the first few iterations, the error at each node may increase or decrease according to the error in the initial estimates of its neighbor. After a few iterations of executing the node sequences in the computation subtree, a global gradient is established that drives the error down across the whole subtree. In the end, each node succeeds in estimating its node location with a 3-centimeter accuracy. The fact that error accumulation is eliminated is due the properties of the computation subtree. As explained in 3.1, collaborative subtrees constitute a constrained configuration of nodes and beacons. This constrained configuration prevents the estimates from going in the wrong direction.

The order of nodes executing in the computation subtree sequence does not need to be pre-specified but it needs to be consistent over successive iterations of the sequence. This entails that the order with which nodes compute their position updates has to be consistent across iterations. One possible way to initiate this distributed computation process is by using Distributed Depth First Search (DDFS) [28]. DDFS search is started at an arbitrary unknown node within the computation tree and its run for two iterations. During the first traversal of the subtree by DDFS, when each node is marked visited, the node computes and broadcasts its location estimate and starts a timer. In the second iteration of DDFS, nodes compute and broadcast their location. At this point the nodes also stop the previously set timer. The time between the two visits denotes the time interval at which each node should recompute and broadcast a new position update. The distributed algorithm for driving the distributed computation process is shown in figure 9.

4.2.2 Localization Accuracy Implications

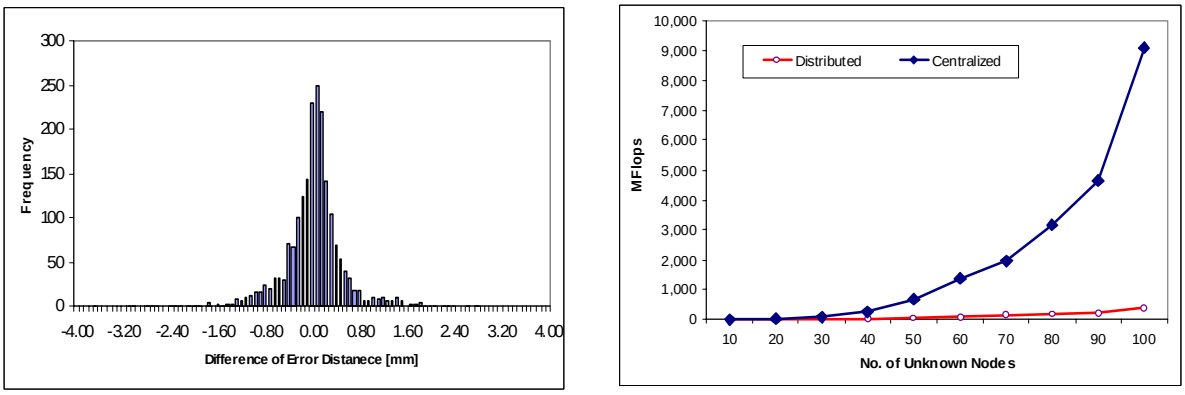
The localized Kalman Filter used in our distributed computation model is an approximation of the Kalman Filter executing in the centralized computation model. Although the

```
Start the algorithm (initiator only!)
visited_u := true
for i:=1 : 2
  for w ∈ Neigh_u
    do begin send ⟨bfs⟩ to w; status_u[w] := cal end

Upon receipt of ⟨bfs⟩ from v:
if not vistited_u(i) then
  begin
    visited_u(i):=true;status_u(i)[v]:=father
    compute new location estimate and broadcast it
    if i=0 t1:=time(); i:=i + 1
    else updatePeriod = time()-t1;
    timer.schedule(updadePeriod)
  end
if status_u(i)[v]=unused then
  begin send ⟨bfs⟩ to v;status_u(i)[w]:= ret end
else if there is a w with status_u(i)[w] :=unused then
  begin send ⟨bfs⟩ to w status_u(i)[w]:=cal end
else if there is a w with status_u(i)[w]=father then
  begin send ⟨bfs⟩ to w end
else (* initiator *) stop

Upon a timeout:
if converged = false or update_needed = true
  begin
    compute a new location estimate and broadcast it
  end
if sqrt((x̂_k)^2 − (x̂_k^-)^2) ≤ Δ
  begin converged := true end
begin timer.schedule(updadePeriod) end

Upon receipt of a updated location broadcast:
  if converged = true
    begin update_needed := true end
```

Figure 9: Distributed computation algorithm driven by DDFS

results of the two methods can be made equivalent if nodes also exchange and update their corresponding covariance matrices, as it is done in [22], in our implementation, we do not communicate the covariance matrices to the neighboring nodes, in the interest of reducing the communication overhead. Before finalizing this design decision, we verified that the results obtained by this approximation do not compromise the overall quality of our estimates. We tested this by comparing the outputs of our centralized and distributed position refinement phases. The tests where run on a test suite of 42 networks of different sizes varying from 10 to 100 nodes. From this comparison we found out that the difference in the results

between the centralized Kalman Filter the distributed approximation we used is very small. Figure 10a depicts the result of the comparison. The mean difference is 0.015 millimeters with a standard deviation of 0.54mm. Based on this result we verified that our distributed approximation of the Kalman Filter does not compromise our computation accuracy.



(a) (b)

Figure 10: a) Accuracy comparison, b) Computation cost comparison

5 EVALUATION

We evaluate the performance of the collaborative multilateration through a set of simulations. The Kalman Filters are implemented in MATALB and they are linked into the ns-2 simulator using the MATLAB compiler and the MATLAB C++ library. The required protocols for communication are implemented in the ns-2 simlulator. Using this simulation setup we carried out a series of experiments on a test suite of 200 different scenarios. Our simulation parameters are set to match the parameters of our experimental node. Each node has an effective radio range of 15 meters and each node can measure distances between its neighbors with the same range as the radio. The measurement noise is modeled as a zero mean gaussian random variable with 20 mm standard deviation, which is based on the characterization results from our ranging system experiments.

The primary goal of our ns-2 implementation is to verify the correct operation of our distributed computation scheme over a wireless ad-hoc network. The distributed algorithm version of the collaborative multilateration is implemented as a routing layer in ns-2. This routing layer also contains a minimal protocol that discovers the two-hop neighborhood of each node, and a forwarding mechanism for forwarding packets with beacons locations as described in section 3.2. These minimal routing requirements are sufficient to form collaborative subtrees in a fully distributed fashion, to obtain the initial estimates and to perform the distributed localization process. At the MAC layer we use a modified version of the IEEE 802.11 protocol to match the specification of our sensor nodes. The radio set to have an effective transmission range of 15 meters with 20Kbps raw data rate.

For the centralized case, Dynamic Source Routing (DSR) /citeJohnson99 was used as the routing protocol, IEEE 802.11 as the MAC and the Kalman Filter was placed at the application layer.

5.1 Computation Cost Comparison

Our first experiment compares the computation overhead between the distributed and centralized computation methods by recording the number of FLOPS consumed by MATLAB to compute the position estimates in each case. The scenarios used for this test have 6 beacons and varying number of unknowns ranging from 10 to 100 nodes. The number of unknowns was used in increments of 10, and the results show the average for 20 scenarios of each type. In all cases the network density is kept constant and each node has an average of 6 neighbors. The cumulative number of MFLOPS for the centralized and distributed implementation are shown in figure 10b. From this result, we found that the computation overhead of the centralized computation model increases fast with the number of unknown nodes. In this particular test, the computation overhead appears to be cubic with the number of nodes. The distributed computation model on the other hand scales linearly with the

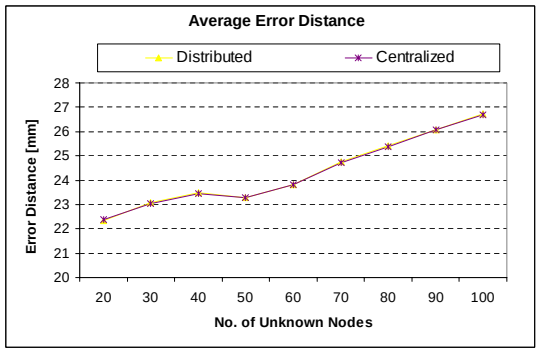


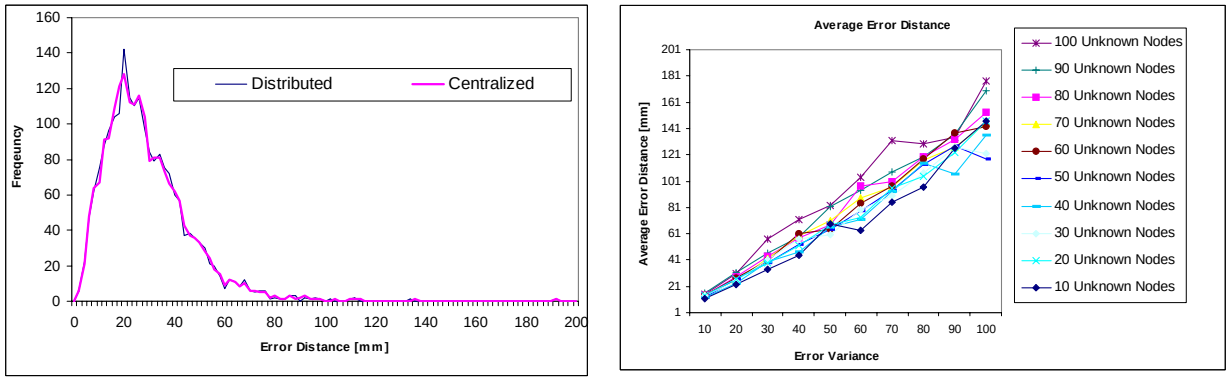
Figure 11: Localization accuracy with increasing number of unknown nodes

number of nodes. The slope for the distributed case in figure 10 is 3.7MFLOPS, meaning that each node spends approximately 3.7MFLOPS to compute an estimate of its location. In our experiments similar trends were observed when varying network density. The computation overhead of the centralized computation model rapidly increases with increasing network density.

From this comparison, we conclude that the distributed computation model is a better choice for computing node locations. Even if computation is performed a central point in the network, using our distributed computation model for computing node locations results in lower latencies than the centralized computation model. This is particularly the case for low cost processors such as the AT91FR4081 processor used on our sensor node.

5.2 Localization Accuracy

To quantify the accuracy of the localization error we applied two tests. The first test evaluates accuracy of the localization process based on the measurement noise parameters of our ultrasonic distance measurement system. Figure 11 shows how the error in the estimates increases as the network scales. The ratio of beacons with respect to the unknowns is kept constant at 20 percent. The error in the estimates increases very gracefully as the network scales.



(a) (b)

Figure 12: a) Estimate error distribution, b) Errors in estimates at different measurement noise levels

Figure 12a shows the cumulative error distribution over all scenarios used in this experiment for both the distributed and centralized cases. In both cases the average error was 27.7 millimeters with a standard deviation of 16 mm.

We observed a similar trend when we computed the error in the estimates at different measurement noise levels. This reflects how the collaborative multilateration would perform with less accurate distance measurement systems as the percentage of beacons descreases. The results of our simulation are shown in figure 12b, are based on different network sizes ranging from 10 unknown nodes to 100 unknown nodes, 20 networks for each set. In all cases the number of beacons is kept constant, with 6 beacons in network. In this particular test we also found that the performance of the distributed version degrades faster in networks of more than 100 nodes. This occurred in cases where the Kalman Filter sequence started by estimating the locations of nodes with accurate initial estimates using neighboring unknown nodes with less accurate initial estimates. One possible approach to mitigate this effect is to assign priorities proportional to the bounding box (computed in section 3.2) dimensions for each node. We expect that if the Kalman Filter computation sequence starts at the nodes

with the largest bounding boxes then the problem mentioned above could be avoided and the convergence will be faster.

5.3 Communication Cost and Convergence Latency

The convergence latency and communication aspects of the collaborative multilateration are more difficult to evaluate because of their dependence on multiple system attributes. In the fully distributed case, convergence latency depends on the available communication bandwidth and the node processing power. Convergence latency also depends on the size of the computation tree. As the number of nodes increases, the sequence of Kalman Filter executions will take longer to complete and more iterations of the sequence are required. The communication pattern is uniform across all the nodes.

When computing at a single point in the network, the convergence latency of the collaborative multilateration primitive is a function of the communication latency for transmitting the packets from each member of the computation subtree and back, and also depends on the power of the central processor. If the computation tree is large, the computation latency will be the dominant component. As an example a network of 100 unknowns and 6 beacons takes approximately 5 to 7 minutes to converge on our Pentium III 700 MHz workstation. Evaluating the communication cost is more complex. In a clustered architecture, the communication cost depends on the cost of electing a cluster head and the cost to propagate the information back and forth from the cluster head.

Figure 13 is a comparison of the communication cost of the collaborative multilateration process in the centralized and distributed case, executed on a network of 44 unknown nodes and 6 beacons. The figure shows the total number of bytes transmitted by each node during the collaborative multilateration process. The average number of bytes transmitted is 4596 for the centralized scheme and 4485 for the distributed scheme. Although on average the

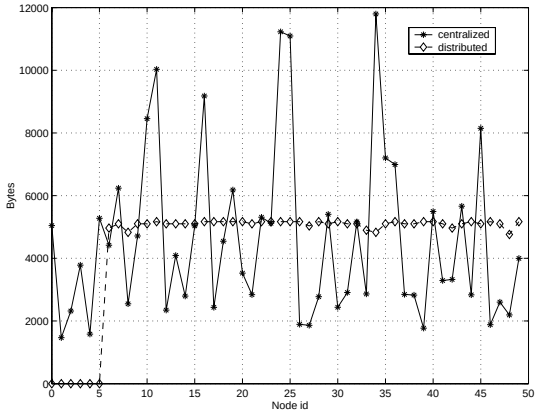


Figure 13: Communication cost on a 50 node network (6 beacons, 44 unknowns)

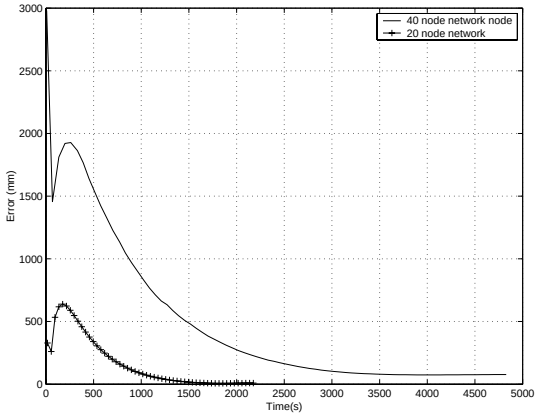


Figure 14: Convergence latency of distributed collaborative multilateration

communication cost is almost the same, the distributed scheme has an even distribution of transmitted bytes. Additionally, a favorable trade of the distributed version on the collaborative multilateration is shown in figure 14. The average convergence latency for two networks sizes of 20 and 40 nodes is shown. During the last part of the process, a lot of computation is spent to achieve a relatively small refinement of the position estimate. At this point, higher-level applications should have the option to decide when to stop the position refinement phase by adjusting the convergence criterion Δ. Finally, we note that this process is robust since the position refinement phase can continue even if some of the nodes fail during the process.

6 Related Work

Node localization has been the topic of active research and many systems have made their appearance in the past few years. Microsoft's RADAR system [1] and USC/ISI GPS-less localization system [4] provide RF based localization. In both cases the unpredictability of wireless links hindered localization accuracy and enforces these systems to rely on preplanned infrastructures. The AT&T Laboratories Active Bats [29], Intersense's Constellation [9] and MIT's Crickets [21] are all infrastructure-based localization systems based on ultrasonic distance measurements. These three systems can perform localization with a few centimeter accuracy but they are all infrastructure based. Some forms of ad-hoc localization have also appeared in the domain of mobile robotics. An example of the latest work in this field is Roumeliotis's Synergetic Localization Scheme [22] in which a group of mobile robots localize themselves by combining a set of inertial measurements and distance measurements using a distributed Kalman Filter. This approach uses a distributed Kalman Filter but has a different setup. The robots start from the same starting point and they keep track of each other by exchanging measurement data and location measurements. This enables the robots to localize themselves with respect to each other. In this scenario, the initial estimates for the Kalman Filter are obtained from odometry measurements, something that is not available in the setup we investigate. Another difference is that this approach does not consider computationally constrained systems in large numbers, which have to utilize measurements and location information over multiple hops.

The positioning algorithm independently developed and proposed in [23] is another ad-hoc localization algorithm that has some similarities to our approach. This positioning algorithm estimates node locations in two phases, Hop-TERRAIN and refinement. In the first phase the Hop-TERRAIN algorithm produces a set of initial node position estimates by measuring the number of hops of each unknown node to a set of anchor points and

using the average hop distance as an approximation to the actual distance from the beacons. In the refinement phase, nodes, compute their locations by performing a set of weighted multilaterations. Niculescu and Nath [19] have also proposed a similar scheme that is based on angle-of-arrival information instead of distance measurements. In this approach error propagation is limited by imposing limits on the number of hops location information is propagated.

Finally, Doherty's [5] convex localization approach describes a method for localizing nodes in an ad-hoc network. This method is based on semi-definite programming and requires rigorous centralized computation so it is not always suitable for low-cost wireless devices. Our approach is different from the previously proposed approaches in the sense that it can estimate locations of nodes that are found multiple hops away from the beacons while avoiding error propagation. We limit error propagation by performing node computation in the context of a constrained node configuration, the computation subtree. Additionally, the proposed approach is flexible, since it can run in many different configurations: fully distributed, locally centralized at a cluster head or purely centralized.

7 CONCLUSIONS AND FUTURE WORK

We have described a node localization scheme for estimating the positions of nodes in ad-hoc networks. Using the collaborative multilateration algorithms described here, nodes that are indirectly connected to beacon nodes can estimate their locations with very high accuracies. Also, with our distributed approach colonies of constrained sensor nodes can collectively solve a global non-linear optimization problem through localized computation and communication. The use of a global gradient for computing a global optimum locally reinforces a distributed computation model with other potential applications in sensor networks. Our simulations have shown that although the computation cost scales linearly with the number

of nodes, the increasing amount of communication suggests that the collaborative multilateration primitive should be applied hierarchically in large networks. In addition to the distributed computation model the collaborative multilateration primitive is an attractive choice for assisting infrastructure based localization systems to better handle obstructions. In this paper we have developed the computational part of the collaborative multilateration. The remaining challenge is to study its feasibility with respect to the physical effects. To this end, as part of our future work we will study the interaction of our algorithms with the physical world on our test bed of Medusa MK-2 nodes.

References

[1] P. Bahl, V. Padmanabhan, RADAR: *An In-Building RF-based User Location and Tracking System*, Proceeding of INFOCOM 2000 Tel Aviv, Israel, March 2000, p775-84, vol 2

[2] Bluesoft Inc *http://www.bluesoft-inc.com*

[3] R. Brown and P. Hwang *Introduction to Signals and Applied Kalman Filtering* Wiley Press 1997

[4] N. Bulusu, J. Heidemann and D. Estrin, *GPS-less Low Cost Outdoor Localization For Very Small Devices*, IEEE Personal Communications Magazine, Special Issue on Networking the Physical World, August 2000

[5] L. Doherty, L. El Ghaoui, K. S. J. Pister, *Convex Position Estimation in Wireless Sensor Networks*, Proceedings of Infocom 2001, Anchorage, AK, April 2001.

[6] D. Estrin, R. Govindan, J. Heidemann, S. Kumar, *Next Century Challenges: Scalable Coordination in Sensor Networks*, Proceedings of the fifth annual international con-

ference on Mobile computing and networking, Seattle, Washington, 1999, Pages: 263 - 270

[7] R. Fontana, S. Gunderson *Ultra Wideband Precision Asset Location System*

[8] roceedings of IEEE Conference on Ultra Wideband Systems and Technologies, May 2002

[9] E. Foxlin, M. Harrington, and G. Pfeiffer *Constellation(tm): A Wide-Range Wireless Motion-Tracking System for Augmented Reality and Virtual Set Applications*, Proceedings of Siggraph 98, Orlando, FL, July 19 - 24, 1998

[10] L. Girod and D. Estrin , *Robust range estimation using acoustic and multimodal sensing* Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2001), Maui, Hawaii, October 2001.

[11] Hexamite http://www.hexamite.com

[12] A. Howard, M. J Mataric and G. S. Sukhatme, *Relaxation on a mesh: a formalism for generalized localization,* Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS01), pages 1055–1060, 2001

[13] Intersense Inc http://www.isense.com

[14] D. Johnson and D. Maltz, *The Dynamic Source Routing for Ad-hoc Wireless Networks* Internet Draft draft.ietf-manet-dsr-03.txt, October 1999

[15] E. Kaplan, *Understanding GPS Principles and Applications* Artech House, 1996

[16] B. Karp, H. T. Kung *GPSR: Greedy Perimeter Stateless Routing for Wireless Networks* Proceedings of International Conference on Mobile Computing and Networking, pp. 243-254, August 6-11, 2000, Boston Massachusetts

[17] J. Li, J Jannotti, D.S J. DeCouto, D. R. Karger and R. Morris, *A Scalable Location Service for Geographic Ad-Hoc Routing*, Proceedings of International Conference on Mobile Communications and Networking, August 11-16, Boston, Massachusetts

[18] S. Meguerdichian, F. Koushanfar, G. Qu, M. Potkonjak, *Exposure In Wireless Ad Hoc Sensor Networks*, International Conference on Mobile Computing and Networking (MobiCom '01),pp. 139-150, Rome, Italy, July 2001..

[19] D. Nicolescu and B. Nath *Ad-Hoc Positioning System* In proceedings of IEEE GlobeCom, November 2001

[20] C. Perkins and P. Bhaghwat *Highly Dynamic Destination Sequenced Distance-Vector Routing* In proceedings of the SIGCOMM 94 Conference on Communication Architectures, Protocols and Applications, pages 234-244, August 1994.

[21] N. Priyantha, A Chakraborthy, and H. Balakrishnan, *The Cricket Location Support System, Proceedings of International* Conference on Mobile Computing and Networking, pp. 32-43, August 6-11, 2000 Boston, MA

[22] Roumeliotis, S.I.; Bekey, G.A. *Synergetic localization for groups of mobile robots*, Proceedings of the 39th IEEE Conference on Decision and Control, Sydney, NSW, Australia, 12-15 Dec. 2000.) Piscataway, NJ, USA: IEEE, 2000. p.3477-82 vol.4. 5 vol. (lxiii+li+5229)

[23] C. Savarese, J. Rabay and K. Langendoen *Robust Positioning Algorithms for Distributed Ad-Hoc Wireless Sensor Networks* USENIX Technical Annual Conference, Monterey, CA, June 2002

[24] A. Savvides, C. C Han and M.B Srivastava *Dynamic Fine-Grained Localization in Ad-Hoc Networks of Sensors* Proceedings of fifth Annual International Conference on Mobile Computing and Networking, Mobicom pp. 166-179, Rome, Italy, July 2001

[25] A. Savvides and M. B. Srivastava *A Distributed Computation Platform for Wireless Embedded Sensing* Proceedings of International Conference on Computer Design, Freiburg, Germany, September 2002

[26] SICK http://www.sick.de

[27] M. Srivastava, R. Muntz and M. Potkoniak *Smart Kindergarten: Sensor-based Wireless Networks for Smart Developmental Problem-solving Environments*, Proceedings of International Conference on Mobile Communications and Networking pp. 132-138, July 16-21, 2001, Rome Italy

[28] G. Tel *Distributed Graph Exploration* Obtained from http://carol.wins.uva.nl/ delaat/netwerken_college/explo.pdf

[29] R. Wand, A. Hopper, V. Falcao and J. Gibbons, *The Active Bat Location System*, ACM Transactions on Information Systems, January 10 1992, pages 91-102

[30] G. Welch and G. Bishop *An Introduction to the Kalman Filter* Available from http://www.cs.unc.edu/ welch/kalman/kalmanIntro.html

[31] BS Rao and HF Durrant-Whyte, *Fully Decentralized algorithm for multisensor Kalman filtering* IEE Proceedings-D, Vol. 138, No.5 September 1991

[32] Y. Xu, J. Heidemann and D. Estrin, *Geography-informed Energy Conservation for Ad Hoc Routing* , Proceedings of the ACM SIGMOBILE 7th Annual International Conference on Mobile Computing and Networking, Rome, Italy, July 2001