Robust Positioning Algorithms for Distributed Ad-Hoc Wireless Sensor Networks

by

Chris Savarese

Research Project

Submitted to the Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, in partial satisfaction of the requirements for the degree of Master of Science, Plan II.

Approval for the Report and Comprehensive Examination:

Committee:

_____

Professor Jan M. Rabaey
Research Advisor

_____

(Date)

_____

Professor Robert W. Broderson
Second Reader

_____

(Date)

Abstract

Distributed algorithms for determining the positions of nodes in an ad-hoc, wireless sensor network are explained in detail. Details regarding the implementation of such algorithms are also discussed. Experimentation is performed on networks containing 400 nodes randomly placed within a square area, and resulting error magnitudes are represented as percentages of each node's radio range. In scenarios with 5% errors in distance measurements, 5% anchor node population (nodes with known locations), and average connectivity levels between neighbors of 7 nodes, the algorithms are shown to have errors less than 33% on average. It is also shown that, given an average connectivity of at least 12 nodes and 10% anchors, the algorithms perform well with up to 40% errors in distance measurements.

Acknowledgements

First and foremost, I have to thank all of the people involved with the Berkeley Wireless Research Center. Without all of their hard work and dedication, this research would never have been possible. Several people were particularly helpful to me. Josie Ammer and Mike Sheets have been offering unconditional support, advice, and friendship from the beginning. Fred Burghardt worked tirelessly to help get several demos working by hard deadlines, all of which were huge successes. Fred also served as my resident C expert on the many occasions when it wasn't just a missing semicolon. Laurent El Ghaoui donated much of his time and insight to reviewing these algorithms and offering several intriguing alternatives. And of course, Jan Rabaey was a fundamental part of this research. Since introducing me to the subject, Jan has helped guide, shape, and motivate my exploration of the positioning problem. His shared excitement and continued support helped make this project the success that it is.

Finally, I would like to extend heart-felt thanks to Koen Langendoen, my partner in research during the summer of 2001. During his summer here, Koen redefined the Refinement algorithm, taking what was only a moderately functional concept and engineering it into the working algorithm presented in this report. Koen deserves much of the credit for the Refinement algorithm as it stands today. Several of the plots included in this work are also his handiwork. Finally, and most important to me, I thank Koen for all of our enlightening discussions and debates about positioning. Working with Koen was truly a pleasure and an education.

This project was funded largely by DARPA under the PAC-C program.

Table of Contents

List of Figures

1 Introduction

1.1 The Positioning Problem within Ad-hoc Sensor Networks

Ad-hoc wireless sensor networks are being developed for use in monitoring a host of environmental characteristics across the area of deployment, such as light, temperature, sound, and many others. Most of these data have the common characteristic that they are useful only when considered in the context of where and when the data was taken from, and so most sensor data will be stamped with position and time information. The very nature of ad-hoc networks, however, makes acquiring this position data quite challenging.

Ad-hoc systems strive to incorporate as few assumptions as possible for characteristics such as the composition of the network, the relative positioning of nodes, and the environment in which the network operates. This calls for robust algorithms that are capable of handling the wide set of possible scenarios left open by so many degrees of freedom. Only two assumptions are absolutely required in a network employing a positioning system that aims at achieving position estimates that are anchored to some three-dimensional global coordinate system. First, all nodes being considered in an instance of the positioning algorithm must be included in the same connected network. Second, there must exist within this network a minimum of four *anchor* nodes. Here, a connected network is a network in which a viable route exists to each node in the network, and an anchor node is a node that is given *a priori* knowledge of its position with respect to some global coordinate system.

A consequence of the ad-hoc nature of sensor networks is the lack of infrastructure inherent to them. With very few exceptions, all nodes are considered equal, making it difficult to rely on centralized computation to solve network wide problems, such as positioning. Thus, distributed algorithms that achieve robustness through iterative propagation of information through a network via multi-hop routing are considered.

These positioning algorithms rely on range measurements to estimate the distances between neighboring nodes. Several techniques can be used to generate these measurements, including time of arrival, angle of arrival, and received signal strength. This algorithm is indifferent to which method is used, except that different methods offer different tradeoffs between accuracy, complexity, cost, and power requirements. Some of these methods generate range measurements with errors as large as ±50% of the measurement, which can result in useless position information if care is not taken. This forms the first of two major challenges in positioning within an ad-hoc space, and will be termed the *range error problem* throughout this report.

The second major challenge behind ad-hoc positioning algorithms, henceforth referred to as the *sparse anchor node proble*m, comes from the need for at least four reference points with known locations in a three-dimensional space in order to uniquely determine the location of an unknown object. Too few reference points results in ambiguities that lead to underdetermined systems of equations. Recalling the assumptions made above, only the anchor nodes will have positioning information at the start of these algorithms, and these anchor nodes will be located randomly throughout an arbitrarily large network. Given limited radio ranges, it is therefore highly unlikely that any randomly selected node in the network will be in direct communication with a sufficient number of reference points to derive its own position estimate.

In response to these two primary obstacles, the algorithms presented are split into two phases: the *Start-up* phase and the *Refinement* phase. The Start-up phase addresses the sparse anchor node problem by cooperatively spreading awareness of the anchor nodes' positions throughout the network, allowing all nodes to arrive at initial position estimates. These initial estimates are not expected to be very accurate, but are useful as rough approximations. The Refinement phase of the algorithm then uses the results of the start-up algorithm to improve upon these initial position estimates. It is here that the range error problem is addressed.

The end goal of these algorithms is to deliver reliable position estimates to other parts of the system that might make use of position information, such as applications or routing algorithms. Such customers of position data naturally prefer consistent

performance, and so the positioning system should favor algorithms that offer consistently moderate error levels over those that sacrifice consistency for the occasional good error level. Also of concern is the cost of the positioning algorithm, both in terms of time and power consumption. Different applications require different specifications in these areas, and so a robust positioning system should include parameters to customize the priority of its task, the overall time available to spend on it, and any algorithm-specific trade-offs between time, accuracy, and power.

1.2 The PicoRadio Project

The PicoRadio project at the Berkeley Wireless Research Center (BWRC) has been the driver for the development of the positioning algorithms discussed in this work, and will be used as a framework in which to motivate discussion. The main goal of the PicoRadio project is to create a ubiquitous, wireless, ad-hoc network of low power sensing and actuating nodes. The primary design guidelines for the project include [14, 1]:

- A flexible, programmable application space
- Ad-hoc, self-configuring networks
- Little or no infrastructure
- Distributed computation models
- Limited radio range per node (about 10m)
- Dense networks connected via a multi-hop routing scheme
- Low power (<100μW)
- Small form factor
- Inexpensive nodes (<$0.50)
- Low data rates (<1kbit/sec/node)

By itself, each individual PicoNode is relatively ineffective and useless, but as a member of a large collection of densely packed nodes (>100 nodes in a 15x15m area, for

example), each PicoNode contributes to a versatile PicoRadio network capable of spreading intelligence across a geographic area in many contexts. PicoRadio enables many services through its abilities to sense and actuate remotely, communicate data across its entire network, and cooperatively process information and computations. Examples of such services include smart environments, various entertainment applications, and other possibilities in automation and robotics [14].

Several of the possible applications of PicoRadio require that the system be easy to install and maintain over very long periods of time. For example, much thought has been given to the possibility of deploying PicoRadio within office environments to enable real-time monitoring and customization of work environments with regard to temperature, air flow, light, and other factors. In order for this to be feasible, the network must be self-configuring to allow for reasonably easy installation, it must have little to no hierarchy to provide robustness against individual node failures, and it must be low power to avoid the need for frequent service. There are other arguments as well, easily justifying the entire bulleted list of attributes given above, and all of these features of the PicoRadio system affect the development of a positioning system that will work with such a long list of variables and restrictions.

1.3 Problem Statement: The Focus of this Work

Ad-hoc sensing networks have a natural need for position information, but are inherently ill-suited for traditional positioning algorithms. The primary goal of this work is to develop a locationing subsystem capable of providing position estimates for individual nodes within a network such as PicoRadio with consistent error margins. The algorithm developed to drive this system should be robust against all degrees of freedom presented by the open-ended application space described above. In the process of developing such a positioning system, the various pitfalls and special cases of deriving position information in low-maintenance, ad-hoc spaces are explored and analyzed, leading to implementation recommendations at both the algorithmic and network levels. Algorithms are judged based on expected error margins in various scenarios, complexity, flexibility, and robustness against special cases.

1.4 Conventions Used Throughout this Report

There are many variables that are relevant to positioning in an ad-hoc network, and this section will quickly set up some conventions to govern the explanations following in the remainder of the report.

There are two types of nodes that will be discussed throughout this work: normal nodes (often just 'nodes'), and anchor nodes. The difference between these two classifications is based on the presence or absence of special hardware or pre-coded data to aid in defining a global position reference. While normal nodes are not equipped with any such abilities, anchor nodes are defined as nodes with *a priori* knowledge of their positions within some global system of reference. This could be accomplished by including GPS receivers, or by hard-coding and immobilizing a small percentage of the nodes in a network, for example. These nodes effectively anchor the network to a system of reference known to entities outside the network. Note that the efforts involved in providing *a prior* position information to a node almost always increase the cost and/or decrease the flexibility of that node, and so it is desirable to have as few anchor nodes as possible. It should also be noted that normal nodes are sometimes referred to as *unknowns* before they develop any position estimates.

Different network configurations make for different challenges from the perspective of a positioning algorithm. To characterize a network, it is important to know:

- The total number of nodes in the network, n
- The number of these n nodes that are anchor nodes, a
- The dimension of the space in which the network resides (i.e. 2-D or 3-D), D
- The size of the space in which the network resides (assumed rectangular), $B_x * B_y * B_z$
- The average range of the radio on each node, given a fixed transmission power and bit-error rate, r
- The variance of the distribution of errors in range measurements (0-mean, AWGN), v

- The average connectivity of individual nodes, *c*
- The average density of nodes in the network, *d*
- The number of iterations Refinement is allowed to perform, *s*

Connectivity and density are functions of various combinations of the other variables listed, and so might be considered redundant. They are used often though, and so they are listed here regardless of their derivability.

Much focus is placed on error magnitudes of position estimates throughout this work. These figures will be given as percentage errors, normalized to *r*, and will refer to the magnitude of the distance between a node's true position and its estimated position. A true position is the absolute physical position of a node within a global reference, and an estimated position is the perceived location of that node, or the best attempt the algorithm can make at guessing the node's true position. Thus, a position error of 25% would imply that there is a Euclidean distance of *0.25*r* units between the position of the node as seen by an omniscient observer and the position of the node as derived by the algorithm.

1.5 Scope and Outline of this Report

The primary focus of this report is on the merits and weaknesses of several algorithms used to position nodes in an ad-hoc sensing network. This work assumes that some other subsystem will provide the range measurements used as inputs to these algorithms, and that these measurements will have some random error distribution associated with them. The algorithms discussed in this work use these range measurements to derive position estimates for each individual node in a network, and then present the resulting coordinates to the rest of the system. This work is not concerned with the trials and tribulations of arriving at these range measurements, nor with the intended use of the coordinates passed out of the positioning system. Some discussion of these bordering issues will be included, but only at highly conceptual levels. Details relevant to implementing and evaluating alternative solutions will be

reserved for the focus on the algorithms used to derive the position estimates from the range measurements.

Chapter 2 will develop the background in the field of positioning within ad-hoc sensing networks. It will cover general triangulation theory, ranging techniques, and other work relevant to ad-hoc positioning systems. Chapter 3 will explain and analyze the algorithms developed to meet the needs of a positioning system for ad-hoc sensor networks, such as PicoRadio. Chapter 4 will cover the experimentation performed to verify the algorithms discussed in Chapter 3. Chapter 5 will focus on implementation details of the algorithms developed in Chapter 3. It will cover such topics as algorithmic cost (in terms of computation, power, and time) and network requirements. Chapter 6 will discuss the algorithms on a more qualitative level, looking at the advantages and disadvantages of various algorithms, possible sources of error, and areas for improvement and future growth. Finally, Chapter 7 will conclude this work.

2 Positioning Background

2.1 Geometric Interpretation

The entire concept of a position is inherently dependent on a predefined frame of reference. In other words, all positions are relative. To say that an object is in the kitchen, for example, is only informative if information about which house contains the kitchen in question is also available. One could then ask which city contains that house, which country contains that city, and so on. Likewise, assigning a set of coordinates to an object is meaningless without knowledge of the coordinate system with which those coordinates are associated. In the context of this work, the concept of a global coordinate system is often taken for granted. Nevertheless, it is assumed that some global coordinate system is available as a reference to which the anchor nodes are tied. The details of this global coordinate system are irrelevant to the positioning algorithms being explored here. In fact, these details are application-specific, in the sense that only the application needs to know if a particular set of coordinates implies that an object is, in fact, in the kitchen. This translation or association between an algorithmic global coordinate system and an environment that humans or other systems might perceive is irrelevant to the positioning algorithms developed here. The only goal of these algorithms is to determine a specific node's location within a given global coordinate system. This is done with triangulation.

Triangulation is a means of locating objects within a coordinate system, and is the foundation for the algorithms presented here. In its most general interpretation, triangulation is a geometric technique that uses edges between objects to determine position. An object is uniquely positioned when at least three reference points are associated with it in a two-dimensional space, thus forming a triangle. These ties between objects are in the form of measured distances and angles. The algorithms in this work make use of a specific implementation of triangulation in which only the measured distances between objects are considered. It is easiest to conceptualize this algorithm by visiting a brief example. Please refer to figure 1:

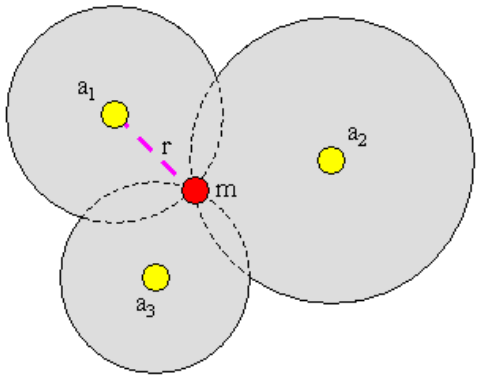


FIGURE 1: GEOMETRIC INTERPRETATION OF TRIANGULATION

Consider a two-dimensional space with an object, *m*, at an unknown position, surrounded by three reference points, a_1, a_2, and a_3, all of which are at known locations. Once *m* has determined the magnitude of its distance, *r*, from a_1, *m* can conclude that it is located somewhere on the perimeter of the circle centered at a_1 with radius *r*. Equivalent knowledge regarding the distance between *m* and a second reference, a_2, implies another circle that intersects the first circle in precisely two points. Finally, *m*'s relationship with a_3 implies a third circle such that all three circles intersect at exactly one point, giving the unique location of *m*. Note that this technique extends to the three-dimensional case by simply adding a fourth reference point. The conceptualization would then start with a sphere associated with the first reference point. This sphere would then be reduced to a circle after the introduction of the second reference point, reducing the situation to the two-dimensional case. This method is similar to the technique used by the Global Positioning System (GPS) [10, 11].

2.2 Ranging Techniques

Modern positioning algorithms rely on one of several forms of raw measurements. Some of the more common techniques for making these measurements are Time of Arrival (ToA), Angle of Arrival (AoA), and Received Signal Strength Indication (RSSI).

ToA is based on the speed of radio wave propagation and the measured time it takes for a radio signal to move between two objects. Combining this information allows the ToA system to estimate the distance between sender and receiver. ToA offers high levels of accuracy, but also requires relatively fast processing capabilities to resolve

timing differences for fine-grained measurements. This problem is amplified over short distances, making ToA a poor choice for a ranging technique for positioning in wireless, ad-hoc sensing networks.

ToA measurements can be combined with acoustic measurements to achieve accuracies of a few percent of the transmission range [6, 18]. Acoustic signals, however, are temperature dependent, require unobstructed line-of-sight, are reliant on directionality, and require additional hardware.

Unlike the previous techniques, which measure distance, AoA techniques make use of antenna arrays to measure the angle at which a signal arrives. Angles can be combined with distance estimates or other angle measurements to derive positions. A major disadvantage of AoA techniques is the hardware required. The antenna arrays are expensive to implement and maintain, making AoA a poor choice for cost-conscious applications (see section 1.2).

RSSI measures the attenuation in radio signal strength between sender and receiver. The power of the radio signal falls off exponentially with distance, and the receiver can measure this attenuation in order to estimate the distance to the sender. Experience has shown, though, that RSSI yields very inaccurate distances [9]. This is the method assumed to be supplying the range estimates to the positioning algorithms discussed in this report.

Although RSSI is assumed to be the method used to generate the range measurements discussed throughout this report, the specific choice of ranging technique has no bearing on the positioning algorithms presented here. Rather, these algorithms are only concerned with the expected error distributions associated with each ranging technique.

2.3 Triangulation

The algorithms presented in this work are based on work done previously by Jan Beutel [1]. Beutel's work made use of the least squares algorithm to derive position estimates from collections of reference points and their associated ranges, and further

measured the effects of overdetermined sets of reference points for improved accuracy in the presence of error.

Triangulation using n reference points and their associated ranges is achieved by first creating the following data structure:

$$\begin{bmatrix} (x_1-u_x)^2+(y_1-u_y)^2+(z_1-u_z)^2 \\ (x_2-u_x)^2+(y_2-u_y)^2+(z_2-u_z)^2 \\ \vdots \\ (x_n-u_x)^2+(y_n-u_y)^2+(z_n-u_z)^2 \end{bmatrix} = \begin{bmatrix} r_1^2 \\ r_2^2 \\ \vdots \\ r_n^2 \end{bmatrix}$$

In the above data structure, (x_i, y_i, z_i) are the three-dimensional coordinates of the i^{th} reference point, (u_x, u_y, u_z) are the coordinates of the unknown node, and r_i is the measured range between the i^{th} reference point and the unknown. This data structure can be linearized by subtracting the last row and performing some minor arithmetic shuffling, resulting in the following relations:

$$Au = b$$

$$A = -2 * \begin{bmatrix} (x_1-x_n) & (y_1-y_n) & (z_1-z_n) \\ (x_2-x_n) & (y_2-y_n) & (z_2-z_n) \\ \vdots & \vdots & \vdots \\ (x_{n-1}-x_n) & (y_{n-1}-y_n) & (z_{n-1}-z_n) \end{bmatrix}$$

$$u = \begin{bmatrix} u_x \\ u_y \\ u_z \end{bmatrix}$$

$$b = \begin{bmatrix} r_1^2 - r_n^2 - x_1^2 + x_n^2 - y_1^2 + y_n^2 - z_1^2 + z_n^2 \\ r_2^2 - r_n^2 - x_2^2 + x_n^2 - y_2^2 + y_n^2 - z_2^2 + z_n^2 \\ \vdots \\ r_{n-1}^2 - r_n^2 - x_{n-1}^2 + x_n^2 - y_{n-1}^2 + y_n^2 - z_{n-1}^2 + z_n^2 \end{bmatrix}$$

The above set of equations can be solved using a traditional least squares algorithm or any variant of it. This work employs the following solution [7]:

$$u = (A'A)^{-1} * A'b$$

As discussed in section 2.1, triangulation in a two-dimensional space can be performed with three or more reference points. In the absence of any errors, these three reference points would provide a perfect solution, and no improvement would be observed from having additional reference points. When errors exist in the references' position or range information, however, the solution will also show some amount of error. The error in the solution can be reduced by adding more reference points, as shown by Beutel.

Errors can also result from situations with poor geometry, or poor constellations. In this context, the constellation refers to the topology of the reference points with respect to the unknown. Beutel shows that the best results can be obtained from constellations with reference points spread out along a perimeter of the unknown, with all reference points being located at similar distances to the unknown. Examples of poor constellations would be sets of reference points arranged in a straight line, or sets containing some very distant and some very close reference points.

Finally, it should be noted that if two reference points appear to be close to each other, systems with less than infinite precision in their arithmetic units could consider their information redundant. Computations involving low numbers of reference points could consequently encounter problems with singularities in the data structures shown above. Singularity failures can also result over time in iterative algorithms if errors compound enough to greatly imbalance the significance of enough reference points.

2.4 Other Research in the Field

The recent survey and taxonomy by Hightower and Borriello provides a general overview of the state-of-the-art in location systems [8]. Few systems for locating sensor nodes in an ad-hoc network are described, however, because of the range error and sparse anchor node problems that are difficult to handle.

An approach that avoids the range error problem altogether is to only use connectivity between nodes. The GPS-less system by Bulusu *et al*. [3] employs a grid of beacon nodes with known locations; each unknown node sets its position to the centroid of the beacon locations it is connected to. The position accuracy is about one-third of the

separation distance between beacons, implying a high beacon density for practical purposes. Doherty *et al*. use the connectivity between nodes to formulate a set of geometric constraints and solve it using convex optimization [5]. The resulting accuracy depends on the fraction of anchor nodes. For example, with 10% anchors the accuracy for unknowns is on the order of the radio range. A serious drawback, which is currently being addressed, is that convex optimization is performed by a single, centralized node. The "DV-hop" approach by Niculescu and Nath, in contrast, is completely ad-hoc and achieves an accuracy of about one-third of the radio range for dense networks [13]. In a first phase anchors flood their location to all nodes in the network. Each unknown node records the position and minimum number of hops to at least three anchors. Whenever an anchor, a_1, infers the position of another anchor, a_2, it computes the distance between them, divides that by the number of hops, and floods this average hop distance into the network. Each unknown uses the average hop distance to convert hop counts to distances, and then performs a triangulation to three or more distant anchors to estimate its own position. DV-hop works well in dense and regular topologies, but for sparse or irregular networks the accuracy degrades to the radio range. DV-hop is very similar to the Hop-TERRAIN algorithm explained in later sections, with the key difference being that DV-hop is considered a complete algorithm, whereas Hop-TERRAIN is designed to work with the Refinement algorithm for improved accuracy, also explained later.

More accurate positions can be obtained by using the range measurements between individual nodes (when the errors are small). When the fraction of anchor nodes is high the "iterative multilateration" method by Savvides *et al*. can be used [18]. Nodes that are connected to at least three anchors compute their position and upgrade to anchor status, allowing additional unknowns to compute their position in the next iteration, etc. Recently a number of approaches have been proposed that require few anchors [12, 13, 17, 4]. They are quite similar and operate as follows. A node measures the distances to its neighbors and then broadcasts this information. This results in each node knowing the distance to its neighbors and some distances between those neighbors. This allows for the construction of partial local maps with relative positions. Adjacent local maps are combined by aligning (mirroring, rotating) the coordinate systems. The known positions of the anchor nodes are used to obtain maps with absolute positions. When three or more

anchors are present in the network a single absolute map results. This style of positioning is not very robust since range errors accumulate when combining the maps.

3 Robust Positioning Algorithms

3.1 The Two-Phase Algorithm Approach

As mentioned in the first chapter of this report, there are two major classes of problems that make positioning within an ad-hoc wireless sensor network challenging: the range error problem, and the sparse anchor node problem. Jan Beutel's work answers the range error problem with redundancy and over-determined systems of equations [1]. The TERRAIN algorithm, described below, provides a clean solution to the sparse anchor node problem. The difficulty, however, is that neither solution works in the presence of both problems.

The redundancy approach set forth by Beutel does a wonderful job of exploiting additional information from extra reference points to average out the errors that occur in the range measurements. He found that, "On average an increase from 3 to 10 nodes used in the solution results in a decrease of the position estimate error by a factor of 4." [1]. Beutel's algorithm assumes, however, that the node being positioned is surrounded by a large number of reference nodes that already know their own locations. In reality, ad-hoc networks such as the ones being considered by this research face the sparse anchor node problem, which specifically says that nodes will *not* be surrounded by other nodes with known locations. This defeats the redundancy approach in its pure form.

Likewise, the TERRAIN algorithm solves the sparse anchor node problem if given perfect range measurements, but breaks down in the presence of any errors in the range measurements. TERRAIN offers a way of spreading the knowledge of a few sparse anchors to the entire network, allowing all nodes to derive accurate position estimates. It is an iterative algorithm though, and it reuses information at a rapid rate. Thus, any errors in raw measurements will be compounded and very quickly explode into such large proportions as to make the final answers derived by nodes utterly useless. As noted earlier, all methods of acquiring raw data have some error associated with them in reality. This is the range error problem, as classified in this report, and it negates the efficacy of the TERRAIN algorithm in its pure form.

To answer the orthogonality of these two separate and conflicting solutions, a two-stage algorithm is proposed. The redundancy approach works great if given initial, accurate positions to work with, and a version of the TERRAIN algorithm can deliver these initial position estimates despite poor initial position awareness. Thus, a version of TERRAIN is slotted into phase 1, termed the *Start-up* phase, and a version of the redundancy approach is slotted into phase 2, called the *Refinement* phase. During the Start-up phase, nodes use a version of TERRAIN to overcome the sparsity of location information and derive initial position estimates. These initial position estimates are expected to be poor (have large errors), but still be accurate enough to provide a launch point for the Refinement phase. The Refinement phase takes these initial position estimates and iteratively improves them, reaching solutions with much higher accuracy over time.

3.2 The Start-Up Algorithm

The purpose of the Start-up phase is to solve the sparse anchor node problem despite errors in the range measurements. This is difficult to do well, however, and in fact, the start-up phase is only responsible for giving nodes rough estimates of their positions in order to generate a first approximation of the topology of the network. This phase of the two-phase algorithm trades off accuracy for consistency on two levels. First, it favors a set of position estimates with consistent levels of error to a set of estimates with some very accurate positions and some extremely poor positions. Second, this stage of the algorithm is interested in consistency across different trials and scenarios. A given start-up algorithm should generate position estimates with errors on roughly the same order of magnitude in almost all scenarios and trials.

This work explores two different start-up algorithms: TERRAIN and Hop-TERRAIN. Although TERRAIN proved to be too inconsistent to use, it is instructive to study it in order to gain insight and appreciation for its successor, Hop-TERRAIN. Both algorithms and their results are explained in the following subsections.

3.2.1 Start-Up Algorithm #1: The TERRAIN Algorithm

TERRAIN stands for *Triangulation via Extended Range and Redundant Associated of Intermediate Nodes*. It is an extension of the ABC algorithm [15] that offers better position estimates. TERRAIN does not, however, offer enough of an improvement over ABC to be useful, since the estimates generated from TERRAIN are too inconsistent. Regardless, this section will briefly outline TERRAIN so as to provide a reference and conceptual starting point for Hop-TERRAIN, discussed in the next section.

The ABC algorithm allows a network of nodes with unknown positions to cooperatively locate themselves with respect to each other, but not with respect to any global reference system. In other words, ABC can be used to create a topologically correct map that is referenced to a coordinate system known only to those nodes in the network. This coordinate system will have a completely random orientation with respect to any global coordinate system. Although rotations could be performed to fix this orientation problem, TERRAIN is layered on top of ABC to achieve this end with less error in the final position estimates.

TERRAIN makes use of the ABC algorithm to incorporate each node into several independent maps; one map for each anchor node. Once a node is included in a sufficient number of maps, it is then able to locate itself with respect to the global coordinate system.

Each anchor node in a network initiates the ABC algorithm, creating *a* independent sets of coordinate systems, each anchored at its respective anchor node. From the perspective of just one of these anchor nodes, the ABC algorithm will propagate through the network, causing each node to be located within the coordinate system for that independent ABC algorithm. A node in the center of the network will eventually acquire *a* sets of coordinates for itself, each relative to one of the maps generated at one of the anchor nodes.

Although each of these positions is referenced to a seemingly random coordinate space, they can be used to derive an estimated distance (or range) to the anchor node

associated with each coordinate system. These distances are the *extended ranges* of TERRAIN, and they are used to artificially extend the visible range of each node so that it can associate itself with each anchor in the network. Once a node has estimated its range to at least one more anchor than the dimension of the space (3 anchors in 2-D, 4 anchors in 3-D), it is able to perform a standard triangulation computation to discover its position in the global space. This procedure is summarized in the following piece of pseudo code:

```
when a positioning packet is received,
        if new anchor or new neighbor then
                if # of neighbors associated with anchor ≥ (dimension of space + 1) then
                        triangulate ABC position
                        compute extended range to anchor
                        broadcast ABC position
                else
                        do nothing.
                if # of extended ranges ≥ (dimension of space + 1) then
                        triangulate global position
                else
                        do nothing.
        else
                do nothing.
```

FIGURE 2: PSEUDO CODE FOR TERRAIN

The following example illustrates the TERRAIN algorithm:

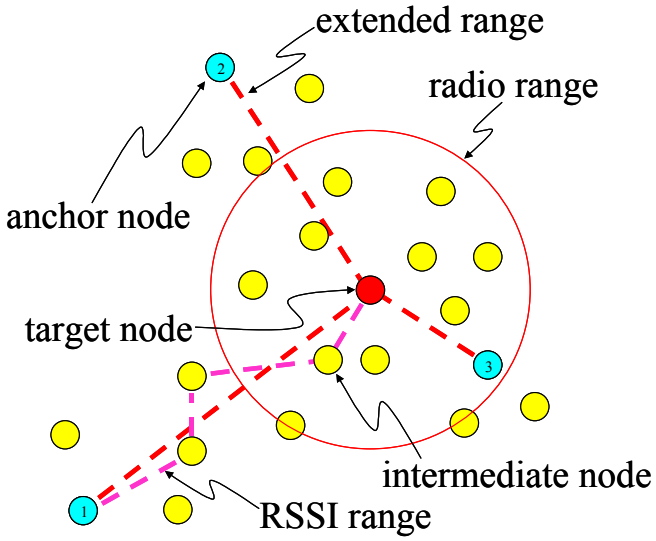


FIGURE 3: THE TERRAIN ALGORITHM

Unfortunately, the TERRAIN algorithm suffers from an unacceptably high tendency to exponentially intensify error levels, such that the final position estimates are too noisy to be useful.

3.2.2 Start-Up Algorithm #2: The Hop-TERRAIN Algorithm

The Hop-TERRAIN algorithm builds off of the TERRAIN concept of extending ranges in order to enable standard triangulation computations. Unlike TERRAIN though, the Hop-TERRAIN algorithm finds the number of routing hops from a node to each of the anchors nodes in a network and then multiplies this hop count by a shared metric (average hop distance) to estimate the range between the node and each anchor. These computed ranges are then used together with the anchor nodes' known positions to perform a triangulation and get the node's estimated position. Hop-TERRAIN realizes advantages over TERRAIN in two ways. First, Hop-TERRAIN does not use the magnitude of the range measured, but rather only checks to see if communication was established. Thus, fluctuations in range accuracy are ignored. Second, Hop-TERRAIN does not iteratively compound errors like TERRAIN does. These advantages help make Hop-TERRAIN much more robust against range errors, yielding more consistent and accurate position estimates on average.

Each of the anchor nodes launches the Hop-TERRAIN algorithm by initiating a broadcast containing its known location and a hop count of 0. All of the one-hop neighbors surrounding the anchor hear this broadcast, record the anchor's position and a hop count of 1, and then perform another broadcast containing the anchor's position and a hop count of 1. Every node that hears this broadcast and did not hear the previous broadcasts will record the anchor's position and a hop count of 2 and then rebroadcast. This process continues until each anchor's position and an associated hop count value have been spread to every node in the network. It is important that nodes receiving these broadcasts search for the smallest number of hops to each anchor. This ensures conformity with the model used to estimate the average distance of a hop, and it also greatly reduces network traffic. One model for estimating the average hop distance between nodes for the entire network is to simply use the maximum radio range of each

node. This simplistic approach is sufficient to generate satisfactory position results, and saves on communication costs relative to more complicated models.

As broadcasts are omni-directional, and will therefore reach nodes behind the broadcasting node, relative to the direction of the flow of information, this algorithm causes nodes to hear many more packets than necessary. In order to prevent an infinite loop of broadcasts, nodes are allowed to broadcast information only if it is not stale to them. In this context, information is stale if it refers to an anchor that the node has already heard from and if the hop count included in the arriving packet is greater than or equal to the hop count stored in memory for this particular anchor. New information will always trigger a broadcast, whereas stale information will never trigger a broadcast. To ensure that no nodes fail to receive information before it is considered stale, anchors should initiate new iterations of the Hop-TERRAIN algorithm at large, fixed time intervals.

Once a node has received data regarding at least three(four) anchor nodes for a network existing in a two(three)-dimensional space, it is able to perform a triangulation to estimate its location. If this node subsequently receives new data after already having performed a triangulation, either a smaller hop count or a new anchor, the node simply performs another triangulation to include the new data. This procedure is summarized in the following piece of pseudo code:

```
when a positioning packet is received,
        if new anchor or lower hop count then
                store (hop count + 1) for this anchor.
                compute estimated range to this anchor.
                broadcast new packet for this anchor.
        else
                do nothing.
        if number of anchors ≥ (dimension of space + 1) then
                triangulate.
        else
                do nothing.
```

FIGURE 4: PSEUDO CODE FOR HOP-TERRAIN

As shown in section 4.2, the Hop-TERRAIN algorithm offers position estimates with extremely consistent error levels.

3.3 The Refinement Algorithm

Given the initial position estimates of Hop-TERRAIN in the Start-up phase, the objective of the Refinement phase is to obtain more accurate positions using the estimated ranges between nodes. Since Refinement must operate in an ad-hoc network, only the distances to the direct (one-hop) neighbors of a node are considered. This limitation allows Refinement to scale to arbitrary network sizes and to operate on low-level networks that do not support multi-hop routing (only a local broadcast is required). Refinement is an iterative algorithm in which the nodes update their positions in a number of steps. At the beginning of each step a node broadcasts its position estimate, receives the positions and corresponding range estimates from its neighbors, and computes a least squares triangulation solution to determine its new position. In many cases the constraints imposed by the distances to the neighboring locations will force the new position towards the true position of the node. When, after a number of iterations, the position update becomes small, Refinement stops and reports the final position. Note that Refinement is by nature an ad-hoc, distributed algorithm. The beauty of Refinement is its simplicity, but that also limits its applicability. In particular, it was *a priori* not clear under what conditions Refinement would converge and how accurate the final solution would be. A number of factors that influence the convergence and accuracy of iterative Refinement are:

- The accuracy of the initial position estimates
- The magnitude of errors in the range estimates
- The average number of neighbors
- The fraction of anchor nodes

When a node has more than three(four) neighbors in a two(three)-dimensional space the induced system of linear equations is over-defined and errors will be averaged out by the least squares solver. For example, the data collected by Beutel [1] shows that

large range errors (standard deviation of 50%) can be tolerated when locating a node surrounded by five or more anchors in a two-dimensional space: the distance between the estimated and true position of the node is less than 5% of the radio range. Despite the positive effects from redundancy it was observed that a straightforward application of Refinement did not converge in a considerable number of "reasonable" cases. Close inspection of the sequence of steps taken under Refinement revealed two important causes:

- Errors propagate fast throughout the whole network. If the network has a diameter *d*, then an error introduced by a node in step *s* has (indirectly) affected every node in the network by step *s*+*d* because of the triangulate-hop-triangulate-hop … pattern.

- Some network topologies are inherently hard, or even impossible to locate. For example, a cluster of *n* nodes (no anchors) connected by a single link to the main network can be simply rotated around the 'entry'-point into the network while keeping the exact same intra-node ranges. Another example is given in figure 5.

To mitigate error propagation the Refinement algorithm was modified to include a confidence metric associated with each node's position. The confidence metrics are used to weigh the equations when solving the system of linear equations. Instead of solving $Au=b$, solve $wAu=wb$, where w is the vector of confidence weights.

Confidence metrics are between 0 and 1. Anchors immediately start off with a confidence value of 1. Unknown nodes start off at a low value (0.1) and may raise their confidence after subsequent Refinement iterations. Whenever a node performs a successful triangulation it sets its confidence level to the average of its neighbors' confidence levels. This will, in general, raise the confidence level. Nodes close to anchors will raise their confidence at the first triangulation, raising in turn the confidence of nodes two hops away from anchors on the next iteration, and so on. Triangulations sometimes fail or the new position is rejected on other grounds. In these cases the confidence is set to 0, so neighbors will not use erroneous information of the inconsistent node in the next iteration. This generally leads to new neighbor positions bringing the faulty node back into a consistent state, allowing it to build its confidence level again. In unfortunate cases a node keeps getting back into an inconsistent state, never converging to a final position or confidence level. To warrant termination the number of position

updates a node may perform is limited to some maximum value. The usage of confidence weights improved the behavior of Refinement greatly: almost all cases converge now, and the accuracy of the positions is also improved considerably.

In addition to confidence metrics, another improvement to Refinement was necessary to handle the second issue of ill-connected groups of nodes. Detecting that a single node is ill-connected is easy: if the number of neighbors is less than three(four) then the node is ill-connected in a two(three)-dimensional space. Detecting that a group of nodes is ill connected, however, is more complicated since some global perspective is necessary. A heuristic is employed that operates in an ad-hoc fashion (no centralized computation), yet is able to detect most ill connected nodes. The underlying premise for the heuristic is that a sound node has *independent* references to at least three(four) anchors. That is, the multi-hop routes to the anchors have no link (edge) in common. For example, node 3 in figure 5, which is taken from [18], meets these criteria and is considered sound.

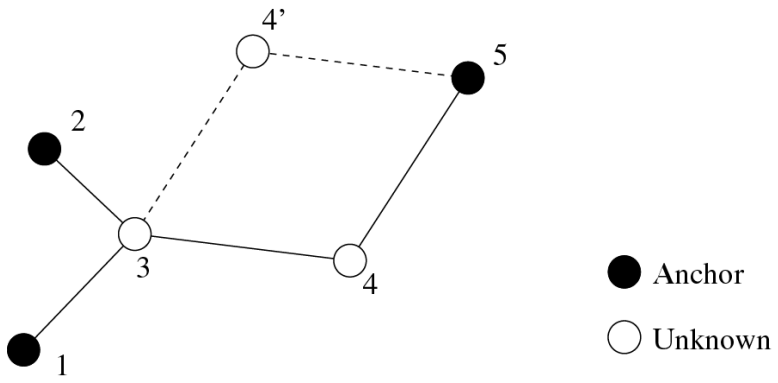


FIGURE 5: ILL-CONNECTED NODES

In the start-up phase, the Hop-TERRAIN algorithm floods the anchor positions through the network and nodes record the hop count of the shortest path to each anchor. Hop-TERRAIN also records the neighbor ID on the shortest path. These IDs are collected in a set of potentially sound neighbors. When the size of this set reaches three(four) a node declares itself sound and may enter the Refinement phase. The neighbors of the sound node add its ID to their sets and may in turn become sound, etc. The end result is that ill-connected nodes will not be able to fill their sets of sound neighbors with enough entries and, therefore, may not participate in the Refinement phase. In the example topology in figure 5, node 3 will become sound, but node 4 will not. Also note that the

more restrictive *participating node* definition by Savvides *et al*. renders both unknown nodes as ill conditioned [18]. Refinement with both modifications (confidence weights, detection of ill-connected nodes) performs quite satisfactorily, as shown by the experiments in section 4.2.

4 Experimentation and Verification of Algorithms

4.1 Simulation Environment

Simulations of TERRAIN, Hop-TERRAIN, and Refinement were performed across three environments: Matlab, a 5000 line C simulator, and the OMNeT++ tools package by András Varga.

Early simulations of TERRAIN were performed in Matlab. All coding was done with Matlab scripts, and all results were summarized with Matlab plots. This environment worked well in the beginning, but proved too slow as simulation sizes grew past 20 node networks.

Noting that Matlab would be too slow to run large numbers of trials on networks containing hundreds of nodes, a C-based simulator program was written. Care was taken to allow for customization of all the necessary parameters, as well as a highly customizable simulation suite mode. This mode of the program allows users to create a suite of simulations to run multiple trials across varying sets of parameters, all of which are programmable at the user interface. Simulations under the C version of the simulator run fast enough to realistically run hundreds of trials on networks with hundreds of nodes over many permutations of different parameters. This feature of the C simulator was essential for determining trends of algorithm performance across many permutations of parameter settings. Results from the C simulator were parsed through simple Matlab scripts for fast, flexible generation of plots.

Finally, a great deal of testing for the Refinement algorithm and some of the experiments for the Hop-TERRAIN algorithm were performed within the OMNeT++ environment. These tools are specifically designed to support large-scale simulations of distributed systems with many nodes, and so simulation of these algorithms fit naturally into their framework.

4.2 Algorithm Performance

In order to evaluate these algorithms, many experiments were run on both Hop-TERRAIN and Refinement. Unless otherwise stated, all data points represent averages over 100 trials in networks containing 400 nodes. The nodes are randomly placed according to a uniform distribution on a 200x200 square, the specified fraction of anchors is randomly selected, and the range between connected nodes is blurred by drawing a random value from a normal distribution with the true range as the mean, and a parametrized standard deviation. The connectivity (average number of neighbors) is controlled by specifying the radio range.

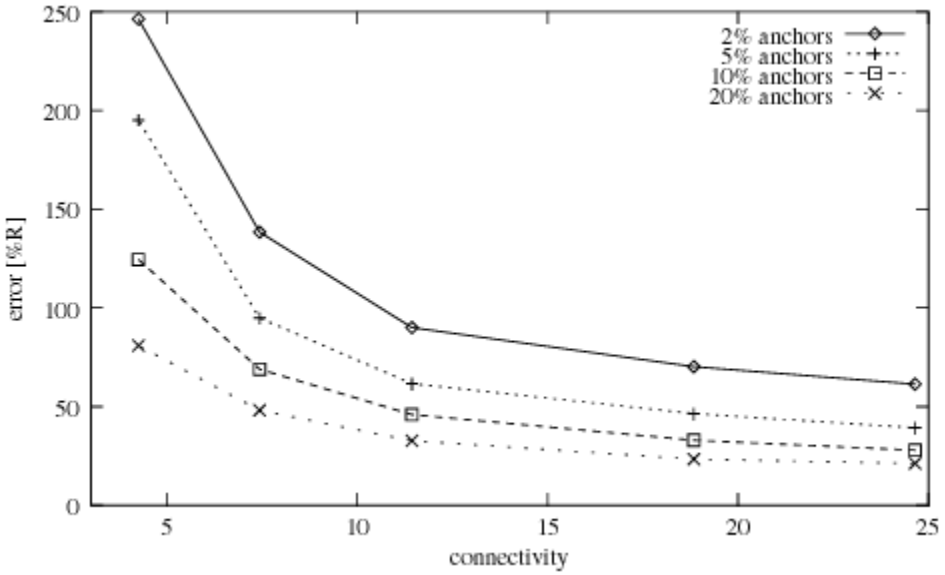


FIGURE 6: AVERAGE POSITION ERROR AFTER HOP-TERRAIN (5% RANGE ERRORS)

Shown in figure 6 is the average performance of the Hop-TERRAIN algorithm as a function of connectivity and anchor population in the presence of 5% range errors[1]. As seen in this plot, position estimates by Hop-TERRAIN have an average accuracy under 100% error in scenarios with at least 5% anchor population and an average connectivity level of 7 nodes or greater. In extreme situations where very few anchors exist and connectivity in the network is very low, Hop-TERRAIN errors reach 250%.

[1] Hop-TERRAIN is shown to be very insensitive to range errors. Range errors are included in this simulation only to further verify that the algorithm operates correctly in the presence of range errors.

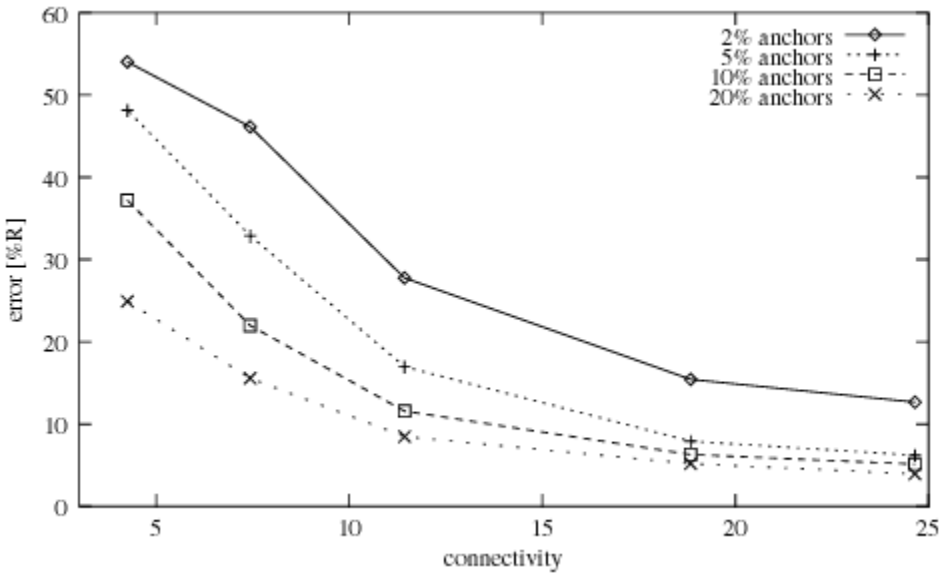


FIGURE 7: AVERAGE POSITION ERROR AFTER REFINEMENT (5% RANGE ERRORS)

Displayed in figure 7 are the results from the same experiment depicted in figure 6, but now the position estimates of Hop-TERRAIN are subsequently processed by the Refinement algorithm. Its shape is similar to that of figure 6, showing relatively consistent error levels of less than 33% in scenarios with at least 5% anchor population and an average connectivity level of 7 nodes or greater. Refinement also has problems with low connectivity and anchor populations, and is shown to climb above 50% position error in these harsh conditions. Overall Refinement improves the accuracy of the position estimates of Hop-TERRAIN by a factor of 3 to 5.

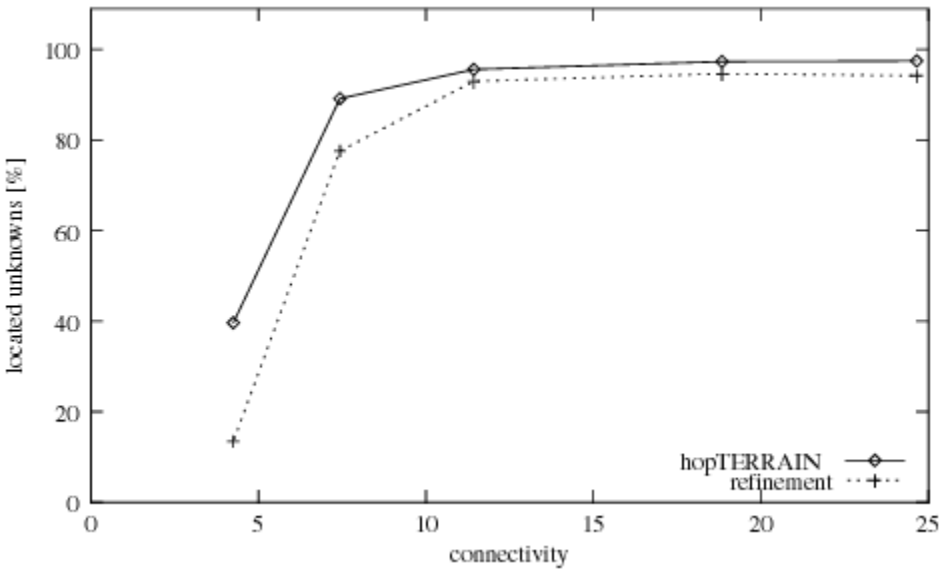


FIGURE 8: FRACTION OF LOCATED NODES (2% ANCHORS, 5% RANGE ERROR)

Figure 7 helps to explain the sharp increases in positioning errors for low anchor populations and sparse networks shown in figures 5 and 6. Figure 7 shows that, as the average connectivity between nodes throughout the network decreases past certain points, both algorithms break down, failing to derive position estimates for large fractions of the network. This is due simply to a lack of sufficient information, and is an impossible consequence of loosely connected networks. It should be noted that the results in figure 8 imply that the reported average position errors for low connectivity levels in figures 5 and 6 have low statistical significance, as these points represent only small fractions of the total network. Nevertheless, the general conclusion to be drawn from figures 5, 6, and 7 is that both Hop-TERRAIN and Refinement perform poorly in networks with average connectivity levels of less than 7 nodes.

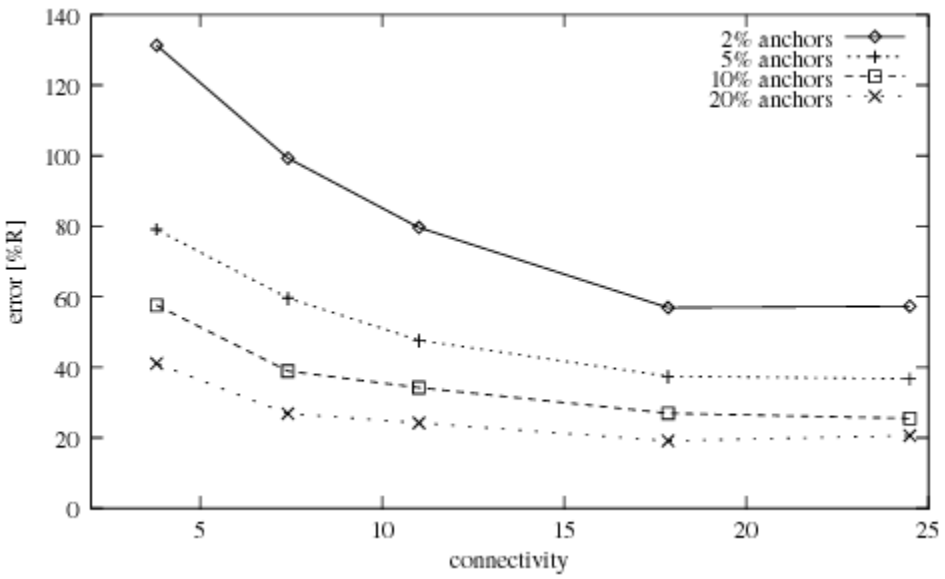


FIGURE 9: AVERAGE POSITION ERROR AFTER HOP-TERRAIN (2D GRID, 5% RANGE ERRORS)

Since connectivity has a pronounced effect on position error it is interesting to ask if other topological characteristics show large effects as well. In the following experiment a fixed grid layout was used instead of randomly placing nodes in a square area according to a uniform distribution. It was found that the grid layout did not result in better performance for the Refinement algorithm, relative to the performance of the Refinement algorithm with random node placement. A plot is not included here because it looks almost identical to figure 7. The performance for Hop-TERRAIN did show a difference though. Figure 8 shows that placing the nodes on a grid dramatically reduces the errors of the Hop-TERRAIN algorithm in the cases where connectivity or anchor node populations are low. For example, with 5% anchors and a connectivity of 8 nodes, the average position error decreases from 95% (random distribution) to 60% (grid). This is likely due to the consistent distances between nodes and the ideal topologies within clusters that result from the grid layout.

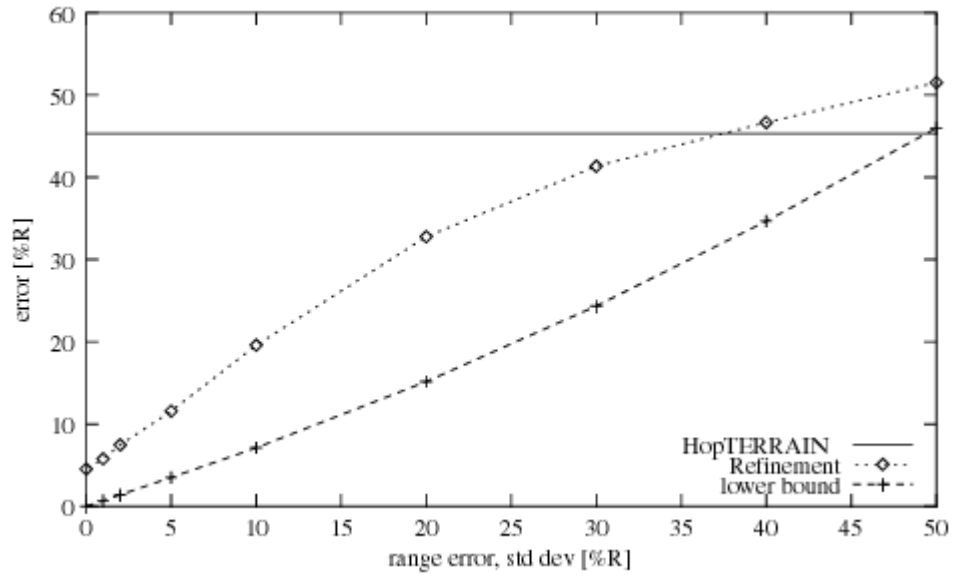


FIGURE 10: RANGE ERROR SENSITIVITY
(10% ANCHORS, CONNECTIVITY OF 12 NODES)

Sensitivity to error levels in range measurements is a major concern for positioning algorithms. Figure 9 shows the results of an experiment in which anchor population and connectivity were held constant at 10% and 12 nodes, respectively, while the average level of error in range measurements was varied. Hop-TERRAIN was found to be almost completely insensitive to range errors. This is a result of the binary nature of the procedure in which routing hops are counted; if nodes can see each other, they pass on incremented hop counts, but at no time do any nodes attempt to measure the actual ranges between them.

Unlike Hop-TERRAIN, Refinement does rely on the range measurements performed between nodes, and figure 10 shows this dependence accordingly. At less than 40% error in the range measurements, on average, Refinement offers improved position estimates over Hop-TERRAIN. The results from Refinement improve steadily as the range errors decrease.

For reference, the best possible position information that can be obtained in each case was determined. A triangulation was performed for each node using the *true* positions of its neighbors and the corresponding erroneous range measurements. The resulting position errors are plotted as the lower bound in figure 10. This shows that there is room for improvement in Refinement.

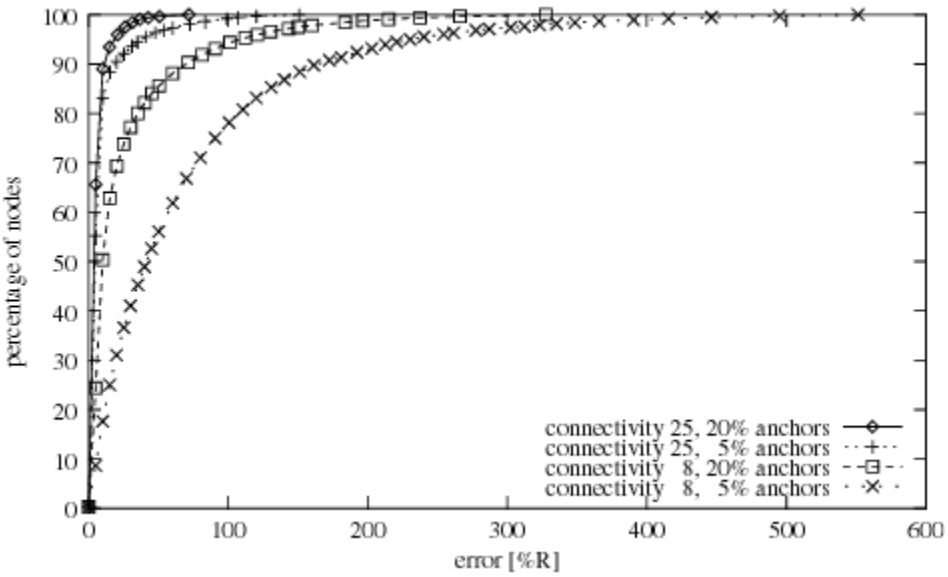
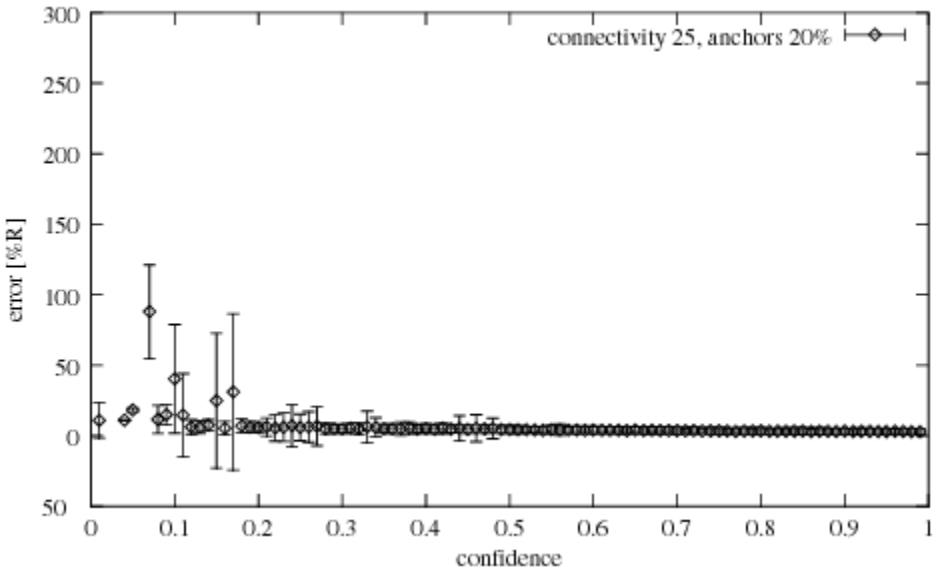


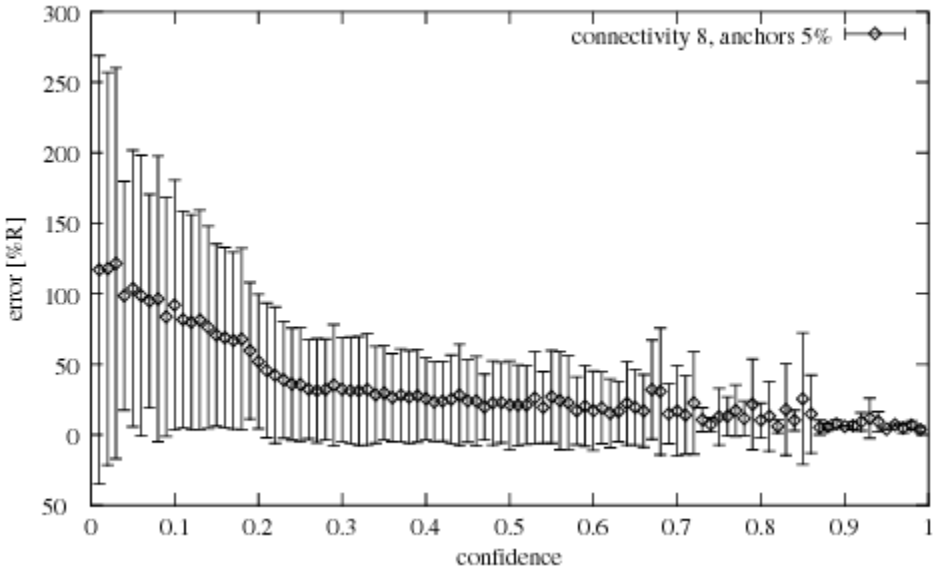
FIGURE 11: CUMULATIVE ERROR DISTRIBUTION (5% RANGE ERRORS)

Up until this point the plots presented report average position errors. Figure 10, in contrast, gives a detailed look at the distribution of the position errors for individual nodes under four different scenarios. Note that the distributions have similar shapes: many nodes with small errors, large tails with outliers.

Refinement's confidence metrics are to some extent capable of pinpointing the outliers. Figure 11 shows the relationship between position error levels and the corresponding confidence values assigned to each node. The data for figure 12 was taken from the best and worst case scenarios from the same experiment used to generate figure 11. As desired, the nodes with higher position errors are assigned lower confidence levels. In the easier case, the confidence are much more reliable than in the more difficult case. The large standard deviations show that confidence levels usually offer good indication of position accuracy at high confidence levels, but can occasionally be misleading. Thus, while confidence metrics can usually be used to derive reasonable faith in position estimates with high confidence levels, the biggest value of using confidences is the improved average positioning error compared to a naive implementation of Refinement without confidences.



(A)



(B)

FIGURE 12: CONFIDENCE VS. POSITION ERROR (AVERAGE AND STANDARD DEVIATION)

Finally, yet another useful way of looking at the distribution of errors over individual nodes is to take their geographical location into account. Figure 12 plots positioning errors as a function of a node's location in the 200x200 testing area. This experiment used 400 randomly placed nodes, an anchor population of 5%, an average connectivity level of 12, and range errors of 5%. The error distribution in figure 13 is quite typical for many scenarios, showing that areas along the edges of the network

lacking a high concentration of anchor nodes are particularly susceptible to high position errors.

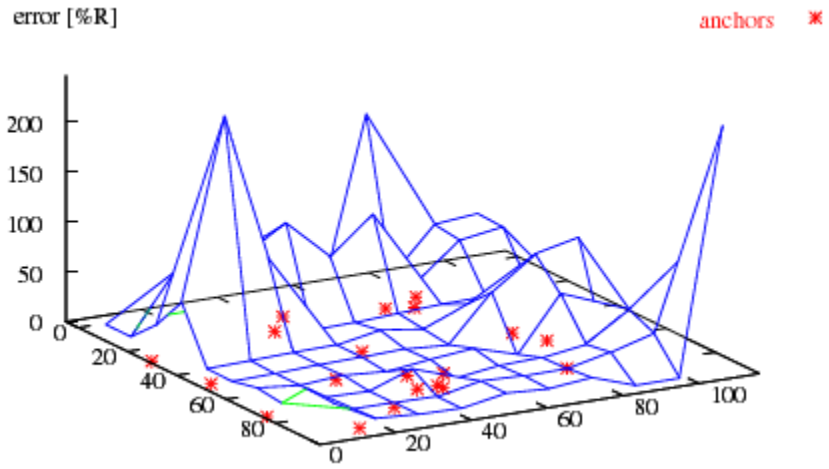


FIGURE 13: GEOGRAPHIC ERROR DISTRIBUTION
(5% ANCHORS, CONNECTIVITY OF 12 NODES, 5% RANGE ERRORS)

5 Considerations for Implementation

Designing an algorithm and implementing an algorithm are two different tasks, and this section of the report attempts to address some of those little details that make moving from concept to implementation a challenging ordeal. Major focus will be placed on the Hop-TERRAIN algorithm, with some attention also given to the Refinement algorithm.

Please refer back to section 1.4 for clarification of variables.

5.1 Power and Computational Requirements

Power is consumed during transmission of packets, reception of packets, and computation. Different nodes in different areas of the network will consume different amounts of power, based on their local network environments, and yet each node has its own isolated power supply. It is important to make estimates of the average power each individual node will spend in performing this algorithm. These estimates will be very rough in nature, as they depend on several assumptions that are very difficult to simulate. All of these assumptions will be explained, as well as the effects they would have on the difference between the estimated and actual costs. In some cases, some assumptions have positive effects on the estimates while others have negative effects, and some rationalization is performed about offsetting these effects to arrive at reliable order-of-magnitude estimates.

5.1.1 Communication Cost

Inherent to the nature of the two algorithms being discussed here is the fact that all communication transactions between neighboring nodes are in the form of broadcasts, rather than directed messages. Thus, all discussions of communication cost will be in terms of the number of broadcast messages a node transmits, BC_T, and the number of broadcast messages a node receives, BC_R.

As discussed earlier in Chapter 3, Hop-TERRAIN begins with every anchor emitting one broadcast packet each, called *hop count packets*. Each anchor's hop count packet contains the x, y, and z coordinates of the respective anchor's location, as well as a hop count value, h_c, initially set to 0. Recalling from Chapter 3, each node that receives one of these hop count packets performs a check to see whether or not the new packet is stale. The main purpose of this check is to greatly reduce the amount of traffic generated by Hop-TERRAIN, thus reducing the power consumed. Provided the packet is not stale, the node will store the information and then broadcast a copy of the packet with $h_c=h_c+1$. This process is allowed to continue until it naturally stops itself when no node is able to generate any information that does not appear stale to all of its neighbors.

The estimate of the number of times an individual node needs to broadcast one of these hop count packets throughout the course of Hop-TERRAIN is affected by two assumptions. First, there is the question of whether every node in the network eventually sees a hop count packet from every anchor in the network. Recalling the definition of a connected network from Chapter 1, this should be the case. In fact, the only ways in which this might not be the case all involve error situations that would either be recovered from over time or result in the conceptual repartitioning of a single network into multiple networks. One example would be the loss of a packet in a critical part of the network, such as an area of low connectivity where only a single node forms the link between two neighborhoods in a network. This situation would be recovered from the next time a wave of broadcasts was initiated. If this critical node were to be removed from the network for whatever reason, then these two neighborhoods would effectively become separate networks. With only slightly higher connectivity levels in a network, however, it is likely that other paths from the anchor in question to the lightly connected nodes would exist. Therefore it is safe to assume that all nodes will hear from all anchors. The resulting cost estimates will only be slightly higher than in reality.

The second assumption affecting the estimate of the number of times an individual node broadcasts a hop count packet addresses the issue of soon-to-be stale information. A situation is shown in figure 14 in which, for whatever reason, information about a particular anchor, a_1, reaches a node, n_{24}, earlier via a longer route than the optimal shortest route.

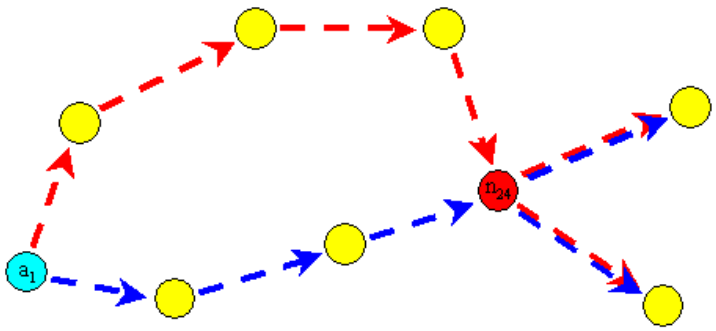


FIGURE 14: UNNECESSARY BROADCASTS

Node n_{24} would dutifully broadcast this sub-optimal hop count packet to its neighbors, causing them to then broadcast the information to all of their neighbors, and so on. When the optimal packet then reaches n_{24}, n_{24} will generate a new broadcast, causing its neighbors to do the same in turn. In other words, because the shortest route packet was delayed, for whatever reason, extra broadcasts were generated in the network, wasting time and power. The likelihood of this sort of situation occurring is based on characteristics of the network such as collision rates, networking protocols, the priority given to handling hop count packets within each node, and the topology of the network. As all of these characteristics are outside the control of positioning subsystem, and therefore dependent on the system in which the positioning subsystem is deployed, it is difficult to model the effect this will have on the number of broadcasts that occur during Hop-TERRAIN. In the interest of getting some first-order estimates of these cost numbers, a very well behaved network will be assumed, and this effect will be ignored for the time being. The effect of ignoring this possibility will be to create estimates that show less cost than occurs in reality.

Ignoring both of the effects discussed above, a simple estimate of the number of times a single node is required to broadcast a hop count packet is given directly by the number of anchors, a. Since each anchor starts a propagation of hop count packets, nodes do not broadcast stale information, and given the above assumptions, each node will hear fresh information from each anchor exactly once, and so will broadcast this information to its neighbors exactly once per anchor. Hence, each node in the network will broadcast an average of a hop count packets.

The average number of times a node receives a broadcast hop count packet is directly related to the average number of transmitted packets. Every time a node sends out a broadcast, all of its neighbors hear it and process the packet in order to determine if the information is stale or fresh. Regardless of the outcome of this check, the cost of receiving the broadcast is still incurred. Each node in the network is considered to have an average of c neighbors. Hence, each node in the network will receive an average of $a*c$ broadcasts of hop count packets.

The communication cost for Refinement is also estimated in terms of the number of transmitted and received broadcasts of packets. Although these packets will not have the same structure as hop count packets, they will be almost identical in size. Refinement communication costs are much easier to measure than those for Hop-TERRAIN because the scope of the Refinement algorithm merely contains each node's one-hop neighbors, rather than the entire network, as is the case with Hop-TERRAIN.

At each iteration of Refinement, each node in the network will broadcast its position information to all of its one-hop neighbors. This equates to the simple estimates of s broadcast transmissions, and $s*c$ broadcast receptions for each node. Optimizations can be implemented to reduce this number, such as only allowing nodes to rebroadcast positions when their information has changed significantly. Such optimizations will reduce this estimate, but are not considered here. Hence, this estimate is slightly pessimistic.

5.1.2 Computational Cost

The computational cost of performing Hop-TERRAIN involves the computation of the extended ranges and the least-squares algorithm. Each node will perform a multiplications in order to compute a extended ranges, one for each of the a anchors in the network. Note that the assumptions made in section 5.1.1 regarding all anchors reaching all nodes are being maintained here. Upon calculating these extended ranges, nodes will then perform least-squares computations in order to estimate their positions, and each least-squares computation has a specific cost associated with it.

Each node will perform a number of least-squares computations that is related to the number of anchors it receives information from. Recall from Chapter 2 that it requires more reference points than there are dimensions in a space to compute an exact location. The first D anchors a node hears from will not result in the node performing a least-squares computation. Each anchor that a node hears from after the D^{th} anchor will, however, cause a new least-squares computation. Thus, the number of least-squares computations each node performs during the course of the Hop-TERRAIN algorithm should be modeled as a-D. For simplicity though, it will be modeled as a. It should be noted that for scaling purposes, nodes do not need to use all a anchors in networks with large anchor populations.

Each least-squares computation carries with it a computational complexity that is proportional to the size of the matrix being operated on. Furthermore, each least-squares computation is actually a two-step process. The data must first be linearized, and then the traditional least-squares formula is applied. Consider a scenario involving a anchors and D dimensions. As explained earlier in Chapter 2, linearization involves the formation of the A and b matrices used in the final least-squares computation. Forming A requires $D*(a-1)$ subtractions. Forming b requires $(a-1)*(2*D+1)$ additions and $(a-1)*2*(D+1)$ multiplications.

Once the data has been linearized, the least-squares algorithm is employed to compute the estimated positions. For a matrix $A \in R^{(a-1)*D}$, the complexity of the least-squares algorithm is $O((a-1)*D^2 + D^3/3)$ [7]. This loosely translates to an equivalent number of flops.

It should be noted that both of the expressions for the linearization and least-squares computation flop counts result in an estimated cost that is higher than the actual cost. This is due to the observation that all but the last of the a-D least-squares operations will involve fewer than a anchors.

Bringing these terms together, each node will perform a multiplication and a least-squares operation for each anchor. Thus, on average, each node will perform $a + a[(a-1)(5D+3) + (a-1)D^2 + D^3/3] = a[1 + (a-1)(D^2+5D+3) + D^3/3]$ flops throughout the course of the Hop-TERRAIN algorithm.

The computational load of Refinement is computed in a manner similar to that used for Hop-TERRAIN. Refinement will cause each node to perform a least-squares operation and to generate new confidence metrics once per iteration of the algorithm, with the algorithm being run for a total of *s* iterations. For Refinement, each least-squares operation will involve all the nodes within each node's one-hop neighborhood. Hence, following the same analysis as for Hop-TERRAIN, each least squares operation will consume $(c-1)(5D+3) + (c-1)D^2 + D^3/3$ flops. Each confidence metric computation requires *c* additions and 1 divide. This gives a total flop count of $s[(c-1)(5D+3) + (c-1)D^2 + D^3/3 + c + 1]$ flops per node for the entire Refinement algorithm.

5.1.3 Power Cost

The results of sections 5.1.1 and 5.1.2 are brought together in figure 15 to give a total power estimate for the complete Hop-TERRAIN/Refinement system of algorithms. Final power estimates are given in terms of BC_T, BC_R, and F, where F is the cost in power of a single flop and is system-specific.

Algorithm	***Communication Cost (power)***	***Computation Cost (power)***
Hop-TERRAIN	$a*BC_T + ac*BC_R$	$a[1+(a-1)(D^2+5D+3)+D^3/3]*F$
Refinement	$s*BC_T + sc*BC_R$	$s[c+1+(c-1)(D^2+5D+3)+D^3/3]*F$
Total	$(a+s)*BC_T + (a+s)c*BC_R$	$\{(a+s)[1+D^3/3]+[a(a-1)+s(c-1)](D^2+5D+3)\}*F$

FIGURE 15: POWER COST OF HOP-TERRAIN AND REFINEMENT

The equations in figure 15 are overly complicated, and can be greatly simplified by observing that D = {2,3}. The same equations are repeated in figure 16, but with D=3, so as to simplify the expressions for computation cost.

Algorithm	***Communication Cost (power)***	***Computation Cost (power)***
Hop-TERRAIN	$a*BC_T + ac*BC_R$	$[27a^2 - 17a]*F$
Refinement	$s*BC_T + sc*BC_R$	$s[28c - 17]*F$
Total	$(a+s)*BC_T + (a+s)c*BC_R$	$[27a^2 - 17a + 28c*s - 17s]*F$

FIGURE 16: SIMPLIFIED POWER COST OF HOP-TERRAIN AND REFINEMENT FOR D=3

5.2 Stabilization Time

A nice feature of the Hop-TERRAIN and Refinement algorithms is their flexibility with respect to time. With one exception, these algorithms have very little dependence on time, and so there is almost no need for any prioritized processor or network time. To this extent, the speed with which the algorithms generate position estimates, and the frequency at which those positions are updated are almost fully parameterizable. The only exception concerns movement of nodes in the network.

If all nodes in a network are stationary, then there is absolutely no need to prioritize processor or network time for any part of either the Hop-TERRAIN or Refinement algorithms. At the other extreme, if all nodes are moving faster than the algorithms could ever possibly complete, then accurate position estimates will never be possible. In a more realistic scenario, most of the nodes in an ad-hoc sensing network are expected to be relatively stationary, and so a relatively low priority level is all that is needed to ensure accurate position estimates.

There is a tradeoff to setting the priority of the positioning algorithms low, of course. As the priority of these algorithms is lowered, the strain on the processor is reduced, but also the time it takes to achieve accurate position estimates or updates is increased. For applications that don't require fast updates on position information, this is not a concern. For those applications that require position estimates within a given time interval, however, the question of the minimum required time for stable position estimates throughout the network becomes relevant.

The minimum time required to finish both algorithms can be expressed in terms of a packet transfer time, t_p. Here t_p is the average time it takes to transfer a single packet between two nodes, including statistics for collisions, processing time, communication latency, and so on. This value, t_p, is assumed to be an average across the entire network. To further clarify this definition for t_p, consider an example where a packet takes seven hops to move from its source node to its destination node. If the value of t_p for that network is perfectly accurate for that particular set of packet transfers, the total time it takes for the packet to be packed and sent from its source and finally unpacked at its

destination would be exactly $7*t_p$. In other words, t_p is such that there is no overlap in processing done at the sending and receiving ends of each hop.

Given t_p, the minimum time required to complete Hop-TERRAIN is simply t_p multiplied by the maximum hop count between the anchor-node pair that has the greatest distance between them. If $B_x = B_y = B$, the maximum distance any such pair could be separated by is $\sqrt{2}B$. If the radio range of the nodes in this network is r, then the total hops required to travel this distance is given by $\frac{\sqrt{2}B}{r}$. Thus, the communication time required is:

$$\frac{\sqrt{2}B * t_p}{r}$$

There are calculations performed at each node as each packet is received. It is assumed that the communication time will be greater than the computation time, and so the computation should be able to be hidden behind the communication. The only exception to this is the final computation performed by the node of that furthest anchor-node pair discussed above. After receiving the packet containing the information about the final anchor, this node will need to perform an extended range computation and a triangulation computation. These computations are detailed above in section 5.1, and it is noted that the actual time required to process them will be dependent on the details of the processor performing them. Let the time required to perform these operations on a given processor be t_H.

Therefore, the minimum time required to perform the Hop-TERRAIN algorithm is given by:

$$\frac{\sqrt{2}B * t_p}{r} + t_H$$

The Refinement algorithm is partitioned into iterations, and the total time required to perform the Refinement algorithm will be a multiple of the time required to perform a single iteration. One iteration of Refinement involves every node in the network receiving a single packet from every node in its one-hop neighborhood, and then every

node in the network performing a single triangulation computation. This is difficult to model, since attempting to have every node in the network broadcast a packet to all of its one-hop neighbors simultaneously would obviously generate nothing more than a very large number of collisions. Assume, therefore, that a cluster of c nodes can be isolated from the rest of the network for the sake of performing this estimate. Also assume that the network protocol allows nodes to engage in full duplex communication. Then the time required for each node to pass its information to every other node in the cluster is given by $c*t_p$, since each node has c neighbors on average. In other words, each node in the cluster has a chance to transmit, and all other nodes in the cluster will listen at that time.

Assuming the nodes in this cluster can perform computations in parallel, the computation time can be modeled as the time required to perform one triangulation computation. As derived above, this requires $(c-1)(5d+3) + (c-1)d^2 + d^3/3$ flops. Let the time required to perform these operations on a given processor be t_R.

Thus, the total time required to perform the Refinement algorithm for s iterations is given by:

$$s\left(c * t_p + t_R\right)$$

It should be noted that these estimated rely very heavily on the ability to compute t_p for a given network. Changing network conditions across time and space for a given network, as well as the random processes that govern collisions, the finer points of any network arbitration protocols, and other factors as well suggest that t_p will be very difficult to model accurately. Nevertheless, this section of the report is intended to give a rough lower bound on the time required to perform these algorithms. Once again, it should be noted that the time required to perform these algorithms is only interesting in those applications that require position information at regular intervals or after a specific latency. Applications that can afford to wait indefinitely for position information, which is assumed to be the majority of cases for the types of networks being considered, will be able to scale the duty cycle of the positioning algorithms down in order to reduce the load on the processor and the network.

5.3 Network Requirements

This section covers several assorted requirements and integration issues regarding the abilities and resources of the network in which these algorithms will be used.

5.3.1 Unique IDs

Whereas some positioning systems require locally or even globally unique IDs, neither Hop-TERRAIN nor Refinement require globally unique IDs, and only Refinement depends on locally unique IDs. This section explores some of the reasons other positioning algorithms require ID information, and it explains how Hop-TERRAIN and Refinement alleviate the need for such information.

Most applications in an ad-hoc sensing network are able to use position for identification. This can be problematic for positioning systems though for two reasons. First, if unique IDs are required by the positioning algorithm in order to establish position estimates, then an additional source of identification other than position is needed. Second, if nodes are mobile, it can become difficult or impossible to differentiate or track nodes with perfect accuracy. For example, if a node carries positioning information from one area of the network to another and does not have any ID other than its own position, it could contaminate the position information of its new neighborhood. Another example is offered by algorithms that require two nodes that cannot see each other to agree on statistics offered by a third mutually connected to both of the original two. If the third node's position is in a state of flux, either because of updates or movement, it becomes impossible for the two original nodes to agree that they are communicating with the same intermediary node. Unique IDs are therefore normally required to identify, track, and differentiate between different nodes.

Fortunately, Hop-TERRAIN has none of these limitations. Refinement will require some ability to track nodes over time though, and so will require at least locally unique IDs. Refinement will not, however, require IDs that are unique across the entire network (globally unique IDs).

Hop-TERRAIN depends on nodes' abilities to pass information about the anchors throughout the network, but it does not rely on information about any particular non-anchor node. Hop-TERRAIN is insensitive to which nodes pass information about the anchors, and so individual node differentiation is not required. The only case in which Hop-TERRAIN requires any differentiation is in the case of the anchors themselves. It is important to be able to distinguish between different anchors so that nodes can identify new anchors, correctly update new hop counts, and accurately filter stale information. Fortunately, anchors have the defining characteristic of having static positions, and so these positions can be used as IDs in the case of the anchors.

There is one subtle problem with relying on the anchors' positions as IDs, and it arises in networks where multiple anchors happen to be located very close to each other. Fortunately, this problem can be avoided by simply placing anchor nodes away from each other, and so this issue is not a large concern for the performance of this algorithm, as explained in the following paragraph.

Depending on the method used to determine the anchor positions, it is possible that some of these anchors could appear to have identical positions, rendering them impossible to distinguish from each other. This would cause the information from all but one of these anchors to be considered stale immediately by all one-hop neighbors, as surrounding nodes will believe they have already heard from that anchor. This would not necessarily cause the system to fail, provided that there were still a sufficient number of anchors outside of the cluster in question to create at least a fully determined system of equations. This condition will always reduce the efficacy of the algorithms though, since less information is being used to compute the estimated positions of the nodes. It should be noted that, even if unique IDs were available to alleviate this problem, such a configuration of anchor nodes would not add any useful information to the system anyway, due to poor geometry. It is therefore recommended that care be taken to ensure that anchors are spaced out from each other sufficiently, relative to the granularity of the positions they are assigned, so that anchors do not appear to be located in identical locations.

Other than the need to identify anchors, Hop-TERRAIN requires no further identification, differentiation, or tracking. Thus, Hop-TERRAIN does not require unique IDs.

Refinement is performed from the perspective of each individual node and is only concerned with the nodes in an immediate one-hop neighborhood. Specifically, each iteration of Refinement is concerned with the position estimates and associated ranges to each node in a one-hop neighborhood. As the networks for which Refinement is designed are not assumed to include synchronization between nodes, it is possible that one or more of the nodes in a neighborhood could change its position estimate during the information collection phase of another node. This could result in multiple representations of a single neighbor in the calculating node's list of reference points, causing undetermined side effects in the results of Refinement. Hence, locally unique IDs are required by Refinement in order to track nodes and ensure that each node is represented only once in a node's list of references.

Other than the need for locally unique IDs for Refinement, the only other requirement is that the information within each transaction is tagged with the local ID of the sender. Each transaction will contain the sender's estimated position and range, and it is necessary to correctly associate each sender's estimated position with the correct estimated range.

In summary, Hop-TERRAIN has no ID requirements, and Refinement only requires locally unique IDs, for the reasons explained above.

5.3.2 Node Density

As shown in section 4.2, both Hop-TERRAIN and Refinement are sensitive to the density of the node population in a network. If nodes are too sparsely connected, the algorithms break down, usually failing to provide position estimates in many cases, and incurring very large levels of error in most other cases.

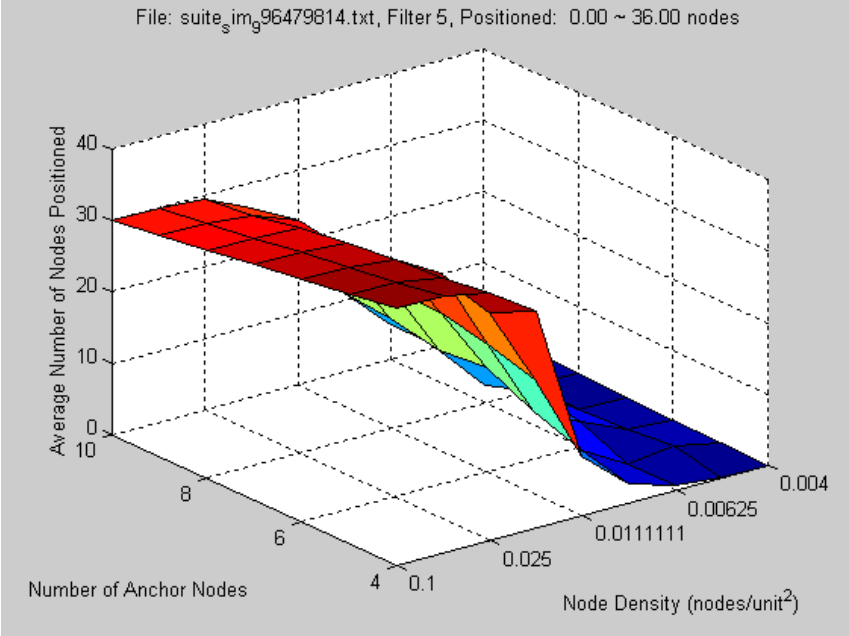


FIGURE 17: QUANTITY OF NODES POSITIONED AS A FUNCTION OF NODE AND ANCHOR DENSITY HOP-TERRAIN, N=40, R=10

The data shown in figure 17 provides another view of the effects of varying density for Hop-TERRAIN. As shown, networks with average densities of 0.025 nodes/unit^2 or higher demonstrate good performance, while networks with lower densities show a sharp cut-off in the number of nodes successfully positioned. Note that a density of 0.025 nodes/unit^2 with r=10 units corresponds to c=7.85 nodes, which agrees with the results in section 4.2.

5.3.3 Anchor Placement

Figure 12 in section 4.2 shows that nodes around the perimeter of a network can suffer large position errors if there are no anchor nodes near the perimeter. In contrast, this same experiment shows that nodes in the middle regions of a network maintain accurate position estimates even in the absence of near-by anchor nodes. It is therefore recommended that some effort be made when installing a network to place anchor nodes around the perimeter of the network.

6 Discussion

6.1 Pros and Cons of Selected Algorithms

Both Hop-TERRAIN and Refinement have their advantages and disadvantages. The following section explores these for both algorithms.

6.1.1 Hop-TERRAIN

Probably the most notable advantage that Hop-TERRAIN offers over TERRAIN and other competitors is its insensitivity to range error, and consequently, its consistency. Hop-TERRAIN achieves this insensitivity to range error by disregarding the measured magnitude of range measurements, and instead noting only a binary condition that checks whether two nodes can see each other. Thus, any fluctuations in the actual range measured, due to calibration differences between nodes, channel characteristics, obstacles, and other sources of error, are ignored by Hop-TERRAIN. The results of this are shown in figure 10, where Hop-TERRAIN shows a remarkable insensitivity to increasing range errors. This insensitivity to range errors naturally results in consistency, as the range errors are the only raw inputs to the system other than the anchor nodes' *a priori* position estimates. An example of Hop-TERRAIN's ability to produce consistent results is shown in figure 18, where Hop-TERRAIN produces average error levels between 70% and 161% for 100 trials.[2]

[2] Each trial consists of a new topology in which all n nodes are randomly placed anew, as well as new applications of noise to the resulting range measurements between pairs of nodes.

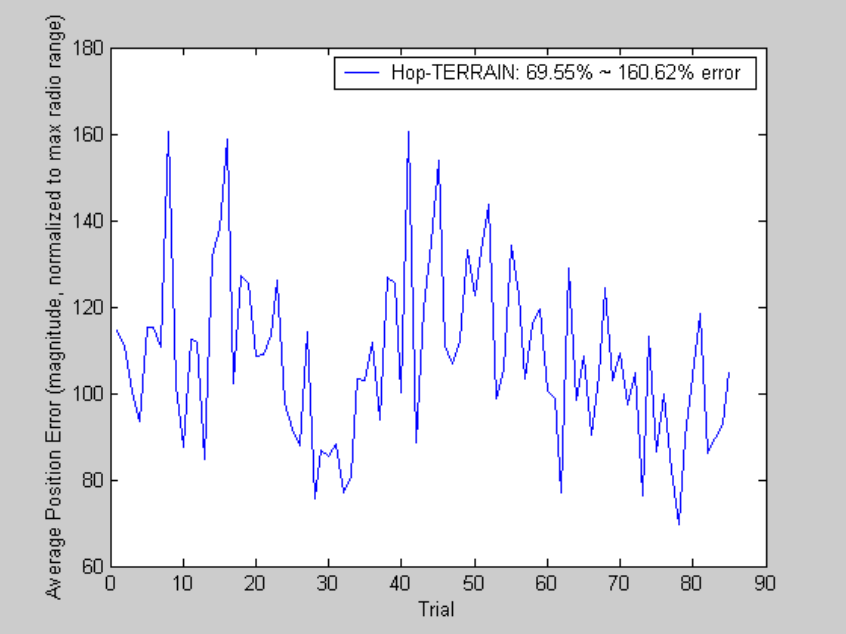


FIGURE 18: CONSISTENCY OF HOP-TERRAIN

The tradeoff for this consistency though, is a reduced level of possible accuracy from Hop-TERRAIN. As shown in previous chapters, Hop-TERRAIN can be expected to consistently produce position estimates for the nodes in a network that average around 100% error, given certain network conditions. Further, Hop-TERRAIN was shown to never produce average errors much greater than 300% error, even in extremely harsh conditions (see Chapter 4). This is in sharp contrast to TERRAIN, as shown below in figure 19. This plot shows Hop-TERRAIN varying from 133% to 309% error across many trials, while TERRAIN ranges from 41% to 2.9E16% error over similar trials. Thus, TERRAIN is able to occasionally achieve better accuracy than Hop-TERRAIN, but Hop-TERRAIN's consistency and substantially lower average error levels make it the more attractive algorithm.

A feature of figure 19 that is not shown in the plot is the success rate of each algorithm at deriving position estimates for each node, regardless of accuracy. On any given pass of these positioning algorithms, some of the nodes in the network may not be able to derive positions. This can occur for certain nodes that are too poorly connected or nodes whose neighboring nodes fail to acquire position estimates. Unique to TERRAIN, this can also sometimes occur from numerical instabilities generated from compounded error. The plots in figure 19 were created by observing all of the trials from a given set of

simulations in which all nodes were successfully positioned. Over the same conditions in an experiment consisting of 2200 trials, Hop-TERRAIN was able to estimate positions for 100% of the nodes for 1078 trials, while TERRAIN successfully positioned 100% of the nodes for only 535 trials.

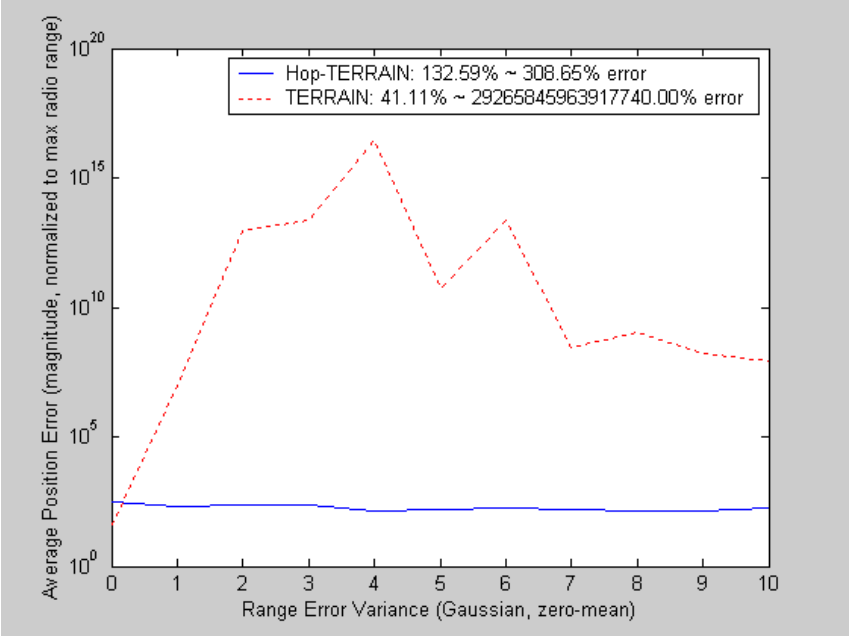


FIGURE 19: RANGE ERROR SENSITIVITY OF HOP-TERRAIN AND TERRAIN N=40, A=4, R=10, B_X=B_Y=30

As a final note regarding figure 19, this experiment was limited to 40 nodes because TERRAIN begins to fail too frequently at network sizes any larger than that. The procedures that occur during TERRAIN cause error to compound across iterations, eventually causing numerical runtime errors in the simulator. More specifically, the errors build up to the point that singular matrices occur within the equations. This leads to the next benefit of Hop-TERRAIN: scalability.

The ability to scale to arbitrarily sized networks is an important criterion for ad-hoc sensor networks. As explained in the previous paragraph, TERRAIN does not scale, due to the compounding of errors and rapid instability of the data structures it uses. Hop-TERRAIN, however, scales well, as shown in figure 20. This experiment tests Hop-TERRAIN's ability to handle networks as large as 1600 nodes, while keeping connectivity fixed at about 12 nodes for all trials. Each data point represents the average error of all nodes for 100 trials. As shown, Hop-TERRAIN remains stable for node

populations up to 1600 nodes. Larger populations were not tested, due to very long simulation times, but it is expected that Hop-TERRAIN will perform well with larger populations as well.

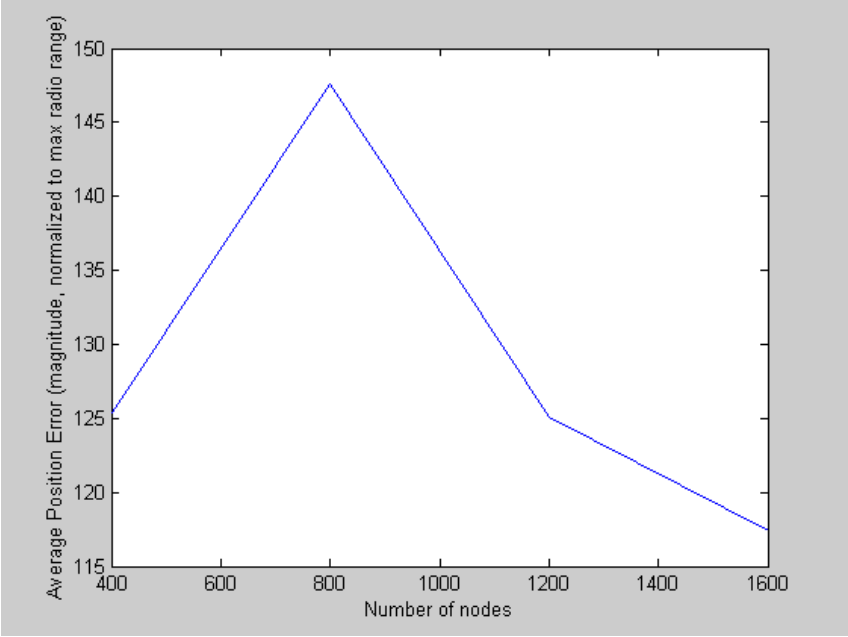


FIGURE 20: SCALABILITY OF HOP-TERRAIN
A=5%, R=10, C=12

Another way in which Hop-TERRAIN scales well is with regard to computation and complexity. The algorithm benefits from being relatively simple, and scaling the network size doesn't change this. Each node retains the same burden of computation regardless of the network size. This is due to the distributed, ad hoc nature of Hop-TERRAIN, which are yet further advantages.

Finally, Hop-TERRAIN offers the programmer a large amount of flexibility in setting the priority for the positioning system. As discussed in section 5.2, the duty cycle of the positioning system can be set arbitrarily low, limited only by the amount of movement in the network.

Hop-TERRAIN is not without flaws though. In addition to sacrificing some accuracy, as mentioned in the paragraphs above, Hop-TERRAIN has two other disadvantages. First, Hop-TERRAIN results in a large amount of network traffic, as detailed in section 5.1.1. The cost of this communication can be eased somewhat by reducing the priority of positioning work, as discussed above.

The second flaw alluded to relates to strange or difficult topologies. Although Hop-TERRAIN appears to be fairly robust against all random scenarios it has been simulated with, it has never been tested with extremely convex networks. An example of such a network is easily developed by taking a concave network and moving a wall deep into the shape from one side of the network, as shown in figure 21.

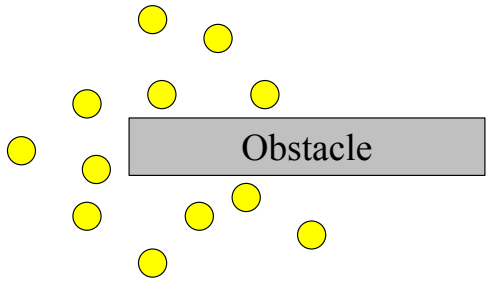


FIGURE 21: CONVEX NETWORK CONFIGURATION

Although Hop-TERRAIN has not been tested with obstacles, there is reason to believe that such a configuration would cause strange side effects in the positioning estimates. The problem arises from the fact that nodes that are physically close to each other, but separated by the obstacle, will receive hop counts that are artificially large from having had to travel around the obstacle. This would distort the estimated ranges used to compute positions, thus distorting the positions themselves [13]. Likely the best solution to this warping effect would be to add more anchor nodes in key locations to mitigate the distortion created by the obstacle.

6.1.2 Refinement

The Refinement algorithm is meant to improve the accuracy of the position estimates generated by Hop-TERRAIN, which it does, as discussed in section 4.2. Refinement is able to achieve this goal under a flexibly duty cycle, like Hop-TERRAIN. Also like Hop-TERRAIN, Refinement scales to arbitrarily sized networks, is distributed and ad-hoc, and offers consistency comparable to that of Hop-TERRAIN, so long as confidence metrics are used. Unfortunately though, Refinement offers only marginal improvement over Hop-TERRAIN in the majority of situations, and at considerable cost. The major drawback to Refinement comes from its dependence on range measurements, as shown in figure 10. This figure shows that situations involving range errors near or

above 40% would not benefit from Refinement, and might even lose accuracy as a result of Refinement. Given unknown channel characteristics, line of sight issues, and unforeseeable interference, it is not unlikely that such large range errors might occur. Hence, Refinement is consistent and worthwhile in situations with range errors substantially lower than 40%, but otherwise it represents wasted energy for most nodes in the network.

6.2 Obstacles to Accuracy

There are three general types of obstacles to accuracy for these algorithms: poor topology, exaggerated range errors, and excessive node mobility.

Stationary obstacles, such as walls, could be a large problem for Hop-TERRAIN due to the falsely inflated hop counts that result. These obstacles artificially create poor topologies, leading to inaccurately estimated extended ranges.

Obstacles could also be a problem for Refinement, especially if radio signals are able to travel through an obstacle, but with some attenuation. In cases like these, the perceived range will be much greater than the actual range, thus worsening the range error even further. Other problems arise for Refinement if obstacles create sections of the network that have low connectivity levels, another example of poor topology. In short, these algorithms work best in an environment free of obstacles. It is expected that they will still operate sufficiently well with some obstacles, but that the error levels across a network of nodes will be less consistent.

In addition to stationary obstacles, some objects may move through a network. For example, a person walking through a room could temporarily distort communication and measurements between nodes. This type of interference would likely be overcome in time, as future passes of the positioning algorithms would not be afflicted by the same disturbance. Nevertheless, environments with chronically high numbers of moving obstacles could result in strange effects in the positioning estimates, for the same reasons discussed in the case of stationary obstacles.

In general, line of sight is very important for both Hop-TERRAIN and Refinement. The algorithms are expected to still work in the presence of some obstacles, but as line of sight between nodes decreases on average across a network, consistency in position accuracy is expected to decline.

Another concern that can contribute to the problem of range errors is the condition of the channel through which the radio signals are passing. Multipath effects, fading patterns, and other forms of interference can have large effects on signal attenuation, thus affecting estimated ranges. In some cases these effects can be large enough to block visibility between two nodes that would otherwise be able to see each other, thus affecting Hop-TERRAIN. In most cases though, channel effects are a concern for Refinement, since they can greatly increase the errors in range measurements.

A final concern to be considered is that of mobile nodes within the network. Mobile nodes present a difficult problem for these algorithms since they may be moving too quickly to establish their own estimates, and more importantly, risk corrupting other neighborhoods of nodes with information that is only relevant to a neighborhood they recently traveled through. Consider the following scenario: If a mobile node measures a hop count to an anchor as being five hops, and then moves to a location in which the same anchor is reported to be twelve hops away, two difficulties arise for Hop-TERRAIN. First, the mobile node will have to start its own positioning process over, and will have to remain stationary long enough to collect enough information from its neighbors before moving to a new, inconsistent location. Second, if the node is moving quickly enough, it is possible that the mobile node would broadcast a hop count of six hops for the anchor in question to its new neighborhood, despite the fact that this new neighborhood is actually an average of twelve hops away from that anchor.

Refinement shares similar problems. If the mobile node moves to a new neighborhood in the middle of collecting data for a new Refinement iteration, it may compile a set of inconsistent data, resulting in a random position estimate. Also, if the mobile node moves to a new neighborhood, it risks contaminating that neighborhood's Refinement calculations if it broadcasts its old position that it knew from its previous location.

If sufficiently few nodes are mobile in a network, or if the speed of movement for the mobile nodes is low enough that iterations of the either Hop-TERRAIN or Refinement appear relatively atomic with respect to network topology, then mobile nodes do not present a large problem. If, however, these conditions are not met, then steps should be taken to reduce the influence that mobile nodes have on the position estimates of their neighbors.

6.3 Areas for Improvement and Future Study

There are two major areas for possible improvement to the Hop-TERRAIN and Refinement algorithms. The first area deals with the numerical engine that drives both of these algorithms: the least squares triangulation method covered in section 2.3. Second, a proposal for future work to attempt to alleviate the shortcomings of Refinement in the presence of high range errors is introduced.

6.3.1 Total Least Squares

Both Hop-TERRAIN and Refinement can be seen as means of manipulating data to better drive the same underlying triangulation algorithm. Both positioning algorithms collect data, combine and manipulate it appropriately, and then use the least squares algorithm to perform triangulation on the data and generate new position estimates. Thus, improving the triangulation model could result in better performance for both algorithms.

As a mathematical model, least squares assumes that errors exist in the b vector, and that the x vector is used to create the best fit from the equation:

$$Ax = b$$

In the context of both Hop-TERRAIN and Refinement, this model therefore assumes that the positions of the reference points in the A matrix are accurate, and only the ranges in the b vector contain errors. This may be sufficiently accurate for Hop-TERRAIN, if the method of obtaining *a priori* positions for the anchor nodes is perfect, but that's a large assumption in any realistic setting. Furthermore, A certainly contains

inaccuracies in the context of Refinement. In general, the assumption that A is without error makes for a poor fit between this mathematical model and its use.

Total least squares is an extension of the least squares algorithm that addresses this problem directly. The total least squares algorithm models errors in both A and b, thus creating a better fit with the positioning problem.

6.3.2 Hop-Refinement

Hop-TERRAIN was adapted from TERRAIN in order to alleviate TERRAIN's sensitivity to range errors. A similar adaptation is recommended for Refinement.

In its current implementation, Refinement instructs a particular node to collect the estimated positions and ranges from all of its one-hop neighbors, and to use this information in a triangulation computation to improve its own position. There are errors in the positions and the ranges used, but the redundancy incurred from correlating information from large numbers of neighbors often manages to generate a better estimate of the node's position. Nevertheless, if the errors present in the positions or ranges of the neighboring nodes are large enough, the accuracy of the new position estimate will degrade.

Rather then remaining sensitive to range errors in this manner, Refinement could be adapted to Hop-Refinement. Hop-Refinement, instead of gathering data from all one-hop neighbors, would seek to refine position estimates by gathering information from subsets of neighbors throughout the network. The ranges to these neighbors would be measured in the same way that Hop-TERRAIN measures ranges: with hop counts and estimated average hop distances. Experimentation would need to be done to determine the appropriate choices of subsets of neighbors to be used, with regard to the number of neighbors per subset, and also the preferred positions of nodes in the subset, relative to the node performing the calculations.

Although this idea would increase communication costs compared to Refinement, these costs could be spread out over time by reducing the duty cycle of the algorithm as needed. This algorithm has not been tested. It is presented here as an idea for possible improvements to the current algorithms, with no evidence of its efficacy.

7 Conclusion

The ability to derive accurate and reliable position estimates for nodes within an ad hoc wireless sensor network is motivated by the intrinsic need for location information to accompany sensor data. Several distributed algorithms for achieving these estimates have been developed and described in detail.

The solution to the positioning problem was divided into two phases, the Start-up phase and the Refinement phase, and algorithms were developed for each. Both the TERRAIN and Hop-TERRAIN algorithms were designed to satisfy the needs of the Start-up phase, but TERRAIN proved unable to overcome the difficulties posed by the range error problem. The Hop-TERRAIN algorithm was adapted from TERRAIN to address exactly this shortcoming. Simulations show that Hop-TERRAIN is able to produce position estimates with average errors of about 100% of each nodes' maximum radio range in the presence of 5% range errors, an average connectivity level of 7 nodes, and an anchor population of 5% of the total nodes in the network. Furthermore, results show that Hop-TERRAIN is insensitive to range errors as great as ±50% of the measured range.

In the same scenario as listed above, Refinement is able to improve the position estimates from Hop-TERRAIN from an average of 100% down to 33%. Unlike Hop-TERRAIN though, Refinement is sensitive to range errors. Experiments show that Refinement offers improved estimates over those generated by Hop-TERRAIN until range errors exceed ±40% of the measured range. Beyond this threshold, Refinement may actually degrade position accuracy, relative to Hop-TERRAIN.

Both Hop-TERRAIN and Refinement scale with arbitrarily sized networks, fit with the model of distributed, ad-hoc systems, and offer consistent accuracy levels in the position estimates they deliver, provided that range errors are below ±40% for Refinement.

Several implementation issues such as algorithmic cost, network requirements, convergence times, and anchor placement were analyzed, and recommendations for future study and possible improvements were discussed.

8 References

[1] J. Beutel, *Geolocation in a PicoRadio Environment*, MS Thesis, ETH Zurich, December, 1999.

[2] N. Bulusu, J. Heidemann, V. Bychkovskiy, and D. Estrin, "Density-Adaptive Beacon Placement Algorithms for Localization in Ad-hoc Wireless Networks," in *IEEE Infocom 200*2, New York, NY, June 2002.

[3] N. Bulusu, J. Heidemann, and D. Estrin, "GPS-less Low-cost Outdoor Localization for Very Small Devices," *IEEE Personal Communication*s, 7(5):28–34, Oct 2000.

[4] S. Capkun, M. Hamdi, and J.-P. Hubaux, "GPS-free Positioning in Mobile Ad-hoc Networks," in Hawaii International Conference on System Sciences (HICSS-34), pages 3481–3490, Maui, Hawaii, January 2001.

[5] L. Doherty, K. Pister, and L. El Ghaoui, "Convex Position Estimation in Wireless Sensor Networks," in IEEE Infocom 2001, Anchorage, AK, April 2001.

[6] L. Girod and D. Estrin, "Robust Range Estimation Using Acoustic and Multimodal Sensing," in IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Maui, Hawaii, October 2001.

[7] G. Golub, *Matrix Computations, 3rd Edition*, The Johns Hopkins University Press, 1996.

[8] J. Hightower and G. Borriello, "Location Systems for Ubiquitous Computing," IEEE Computer, 34(8):57–66, Aug 2001.

[9] J. Hightower, R. Want, and G. Borriello, "SpotON: An Indoor 3D Location Sensing Technology Based on RF Signal Strength," UW CSE 00-02-02, University of Washington, Department of Computer Science and Engineering, Seattle, WA, February 2000.

[10] J. Juang, "On GPS Positioning and Integrity Monitoring," IEEE Transactions on Aerospace and Electronic Systems, Vol. 36, No. 1, January 2000.

[11] T. Logsdon, *The Navstar Global Positioning System*, Van Nostrand Reinhold, 1999.

[12] R. Mukai, R. Hudson, G. Pottie, and K. Yao, "A Protocol for Distributed Node Location," IEEE Communication Letters, to be published.

[13] D. Niculescu and B. Nath, "Ad-hoc Positioning system," In IEEE GlobeCom, November 2001.

[14] J. Rabaey, J. Ammer, T. Karalar, S. Li, B. Otis, M. Sheets, and T. Tuan, "PicoRadios for Wireless Sensor Networks – The Next Challenge in Ultra-Low Power Design," Proceedings of the International Solid-State Circuits Conference, San Francisco, CA, February 3-7, 2002.

[15] C. Savarese, J. Rabaey, and J. Beutel, “Locationing in Distributed Ad-hoc Wireless Sensor Networks,” in IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP), pages 2037–2040, Salt Lake City, UT, May 2001.

[16] C. Savarese, J. Rabaey, and K. Langendoen, “Robust Positioning Algorithms for Distributed Ad-Hoc Wireless Sensor Networks,” To be published in the proceedings of the USENIX Annual Technical Conference, 2002.

[17] A. Savvides, C.-C. Han, and M. Srivastava, “Dynamic Fine-Grained Localization in Ad-hoc Networks of Sensors,” in 7th ACM Int. Conf. on Mobile Computing and Networking (Mobicom), pages 166–179, Rome, Italy, July 2001.

[18] A. Varga, “The OMNeT++ Discrete Event Simulation System,” in European Simulation Multiconference (ESM’2001), Prague, Czech Republic, June 2001.