P - 1639

LOCALIZED OPTIMIZATION PROCEDURE FOR LOCATION DISCOVERY

A PROJECT REPORT

Submitted by

ANUSHA.M	71202205002
HARINY.A.B	71202205015

in partial fulfillment for the award of the degree

of

BACHELOR OF TECHNOLOGY

IN

INFORMATION TECHNOLOGY

KUMARAGURU COLLEGE OF TECHNOLOGY, COIMBATORE.

ANNA UNIVERSITY: CHENNAI 600 025

APRIL 2006

ANNA UNIVERSITY: CHENNAI 600 025

BONAFIDE CERTIFICATE

ertified that this project report "**LOCALIZED OPTIMIZATION ROCEDURE FOR LOCATION DISCOVERY**" is the bonafide work of **ANUSHA.M** and **HARINY.A.B** " who carried out the project work under my upervision.

[illegible]

IGNATURE

Dr.G.Gopalsamy

HEAD OF THE DEPARTMENT

Dept of Information Technology,

Kumaraguru College of Technology,

Coimbatore – 641006.

[illegible] 24/4/06

SIGNATURE

Mrs.L.S.Jayashree

Assistant Professor

Dept of Computer science and Engineering,

Kumaraguru College of Technology,

Coimbatore – 641006.

The candidates with University Register Nos.71202205002,71202205015, were examined by us in the project viva-voce examination held on 26.4.06.

[illegible]

INTERNAL EXAMINER

[illegible] 26/4/06

EXTERNAL EXAMINER

ACKNOWLEDGEMENT

ACKNOWLEDGEMENT

We express our profound gratitude to **Dr. K. K. Padmanabhan**, B.Sc. (Engg)., M.Tech., Ph.D., Principal, Kumaraguru College of Technology, Coimbatore, for permitting us to undergo a project work.

We are greatly indebted to **Dr. G. Gopalsamy**, Ph.D., Professor and Head of the Department of Information Technology, for the immense support he has provided us throughout our project.

We extend our sincere thanks to our project coordinator, **Prof.K.R.Baskaran**, B.E.,M.S., Assistant Professor, Department of Information Technology, for his constant support and encouragement.

We are much obliged to express our sincere thanks and gratitude to our beloved guide **Mrs. L .S. Jayashree,** Assistant Professor, Department of computer science and Engineering, who gave the initial idea of the project and guidance to carry out every further step. We also thank our co-guide, **Ms. M. Lavanya** for her valuable suggestions, and encouragement which has enabled us to complete the project successfully.

We gratefully thank our Class Advisor **Mrs. N. Suganthi**, Senior Lecturer, Department of Information Technology, for extending her most appreciative and timely help to us.

We also thank all the teaching and non-teaching staffs members of the Department of Information Technology for all their encouragement and moral support.

We also extend our heartiest thanks to all our friends for their continuous help and encouragement throughout the course of study.

DECLARATION

We,

Anusha.M 71202205002

Hariny.A.B 71202205015

declare that the project entitled **"Localized Optimization Procedure For Location Discovery"**, submitted in partial fulfillment to Anna University as the project work of Bachelor Of Technology (Information Technology) Degree, is a record of original work done by us under the supervision and guidance of Mrs.L.S.Jayashree, Assistant Professor and co-guide Ms.M.Lavanya, Lecturer, Department of Information Technology, Kumaraguru College Of Technology, Coimbatore.

Place : Coimbatore
Date : 24·4·06

[illegible]
[Anusha.M]

A·B·Hariny
[Hariny.A.B]

Project Guided by

[illegible] 24/4/06

[Mrs. L.S. Jayashree]

[illegible]

[Ms. M. Lavanya]

[illegible]

Head of the Department
[Dr. G. Gopalsamy]

ABSTRACT

ABSTRACT

Wireless Ad-hoc Sensor Networks (WASNs) consist of a number of sensor nodes each equipped with a certain number of sensors and some amount of communication, storage and processing resources. The development of practical, localized algorithms is probably the most needed and most challenging task in Wireless Ad-hoc Sensor Networks. Localized algorithms are a special type of distributed algorithms where only a subset of nodes in the WASN participates in sensing, communication, and computation.

Localization is the process of determining the spatial co-ordinates of sensor nodes based on the set of already available GPS equipped nodes. These GPS equipped know their positions and are called as beacons. The nodes that are yet to resolve their geographical co-ordinates are called as orphans. The Localization techniques are classified as range-based and range-free schemes.

In this project, we use TRILATERATION , a centroid based range-free scheme using which the nodes determine their location with help of three beacons in their communication range. It is an iterative process, which terminates when no more nodes can estimate their location using the beacons.

Localization quality takes into account three parameters namely the mean error in determining the location, the number of rounds and the number of orphans. The Location quality obtained by trilateration is further improved using HEAP-MAX algorithm which identifies the candidate points where extra beacons need to be deployed.

We also prove the scalability of the algorithm by means of testing it for varying node densities.

CONTENTS

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF GRAPHS

LIST OF ABBREVIATIONS

1.	WASN	-	Wireless Ad-hoc Sensor Networks
2.	ToA	-	Time of Arrival
3.	TDoA	-	Time Distance of Arrival
4.	AoA	-	Angle of Arrival
5.	RSSI	-	Received Signal Strength Indicator
6.	BPRN	-	Beacons Per Radio Neighborhood
7.	ARLE	-	Average Relative Error

INTRODUCTION

1. INTRODUCTION

In sensor networks, nodes are deployed into an unplanned infrastructure where there is no a-priori knowledge of location. The problem of estimating spatial-coordinates of the node is referred to as localization. An immediate solution that comes to mind is GPS or the Global Positioning System. However, there are some strong factors against the usage of GPS. For one, GPS can work only outdoors. Secondly, GPS receivers are expensive and not suitable in the construction of small cheap sensor nodes. A third factor is that it cannot work in the presence of any obstruction like dense foliage etc. Thus, sensor nodes would need to have other means of establishing their positions and organizing themselves into a coordinate system without relying on an existing infrastructure. The main practical objective is to locate each node as accurately as possible with a given information with a certain amount of error about the distances between a subset of nodes.

1.1. Existing System

Localization of a node has been done using the centralized approaches. However, in centralized approach, the nodes that need to estimate their location has to contact the central server. The amount of communication and power involved in computing their location is very high. But, in sensor networks, power consumption must be as minimum as possible. The localization protocols used to estimate a node's location falls in to two categories: Range-based and Range-free. The former is defined by protocols that use absolute point-to-point distance estimates or angle estimates for calculating location. The latter makes no assumption about the availability or validity of such information. The already available range-based schemes like Time of Arrival (ToA), Time Difference of Arrival (TDoA), Angle of Arrival (AoA) are cost consuming.

All these techniques make use of the Received Signal Strength Indicator (RSSI) measurements. So, additional power is consumed in transmitting and receiving the signal back. While solutions based on RSSI have demonstrated their efficacy in simulation based on signal strength, propagation terms and fading models, they are not suitable for resource-constrained environments.

Limitations of the Existing System

(i) Relative amount of communication are much higher .

(ii) Power is probably the single most important limiting factor.

(iii) The range-based schemes are cost consuming.

(iv) Time consumption for computing the location is more.

1.2. Proposed system

The proposed system is based on localized algorithms that are range-free in nature. The localized algorithm has the following five components. Data acquisition mechanism facilitates which sensed data is obtained from which node. The optimization mechanism provides a partial or complete solution to the targeted task. Search expansion rules indicate which nodes are best to contact next. Bounding conditions indicate which nodes should not be considered further, since information that they have is irrelevant for the final solution. Finally, termination criteria indicate when search expansion and optimization mechanism can be halted.

Trilateration, a range-free scheme used in this system, follows a proximity-based approach. It is a method by which a sensor node estimates its location based on three other nodes which already know their location. These nodes that already know their location information are called "Beacons" or

"Anchor nodes". The localization here does not involve much communication and hence cost effective.

The location estimates obtained are further optimized using the HEAP-MAX algorithm implementation. Here, the whole terrain where the sensors are deployed is divided into a number of small squares and the location error is simulated at predefined points. The points which have the highest measured location error are sorted and additional beacons are deployed according to the degree of localization quality that is required by the user.

Advantages of the proposed System

(i) The relative amount of communication is lesser.

(ii) Power consumption is less as compared to the range-based schemes.

(iii) The range-free schemes are best suited for non-critical applications.

(iv) Relatively less time for collecting and manipulating location estimates since the node information is collected locally.

SYSTEM REQUIREMENT ANALYSIS

2. SYSTEM REQUIREMENT ANALYSIS

2.1. Project plan

The requirement phase deals with the sequence of activities in producing the Software Requirement Specification (SRS) document. It must be ensured that all the requirements of the software are elicited and analyzed. In other words the needs of the system are identified.

In the problem analysis phase, the current system is analyzed by the study of the existing materials, and the changes to be made in the proposed system are decided upon. A clear understanding of the needs of the system must be framed. Analysis leads to actual to actual specification.

Designing aims at how to satisfy the needs of the system. The different modules of the system and the interaction between these modules to produce the desired functionality are identified. During detailed design, the internal logic of each module and their algorithmic design is specified. The major and important decisions are made in this phase.

Coding is the process of translating the design into code. The code developed must be easy to understand. Well-written code can reduce testing and maintenance effort.

Testing is done to check if the requirements of the user are met. It involves in checking the functionality of each module as per design document.

2.2. Software Requirement Specification

2.2.1. Purpose

The purpose of our project is to enable the nodes to estimate their location in a cost effective manner by the help of a localized range free location discovery technique.

2.2.2. Scope

The nodes estimate their location using Trilateration, which is a range-free approach and The estimated locations are optimized using HEAP-MAX algorithm.

2.2.3. Product Overview and Summary

This simulation is useful for the any event detection applications that deploy sensor nodes for monitoring. This simulation gives an idea to the deployer as to how the nodes can be deployed efficiently, so that the location where the event occurs can be detected appropriately.

2.2.4. Development and Operating Environment

2.2.4.1. Software Constraints

Operating System	:	Windows/Ms Dos
Language	:	C++

2.2.4.2. Hardware Constraints

Processor	:	Intel Pentium
RAM	:	128 MB

2.2.5. Functional Specifications

Our project consists of the following modules:

- Initialization
- Location Information Exchange
- Location Discovery
- Termination
- Optimization Using HEAP-MAX

Initialization

A. Checking for node status

- GPS Equipped (Beacon)
- Beacon
- Orphan

B. Broadcasting status information

Location Information Exchange

A. Recording Information from all neighbors

B. Sorting Orphans and Beacons

Location Discovery

In this Phase, a common node that does not know its location estimates its location based on three beacons using trilateration procedure.

Termination

The location discovery phase continues till the common nodes can estimate their location by three beacons. Some nodes cannot estimate their locations either because they do not have three beacons or they may have no connectivity to their neighbors. Termination phase identifies this state when nodes cannot estimate their location and stops the previous phase

Optimization using HEAP-MAX:

The estimated location of a node based on the beacons will never be accurate. So some optimization is required. The HEAP-MAX algorithm divides the terrain in to a number of squares and simulates the location error at different points in the terrain. The points which have the highest observed location error are identified and Additional beacons are added according to the quality of localization required.

2.2.6. General Assumptions

The following assumptions are made while simulating the project

- The sensor nodes are deployed in random fashion
- The effect of noise in transmission is negligible
- The sensor nodes deployed are all in good working conditions
- All the messages are transmitted and received correctly
- The beacons are also randomly deployed

2.2.7. User characteristics

The users of the system should have a detailed idea of the environment in which the sensors are to be deployed. Based on the application the deployment will vary. The user should also have an idea about the windows and sensor networks to understand the results of the simulation.

2.2.8. Performance Constraints:

The simulation works well for a low density or medium density sensor networks .When the number of nodes is beyond hundred it is difficult for the graphic driver to load and execute the input and output commands from the input and output ports. Therefore, a little delay is encountered. When the number of nodes is very less, there is not enough connectivity between the nodes and all of them remain as orphans.

2.2.9. Software Constraints:

The system will run only under the windows/Ms-Dos environment and it needs the Turbo C++ to be installed in the system of the user

SYSTEM STUDY

3. SYSTEM STUDY

3.1. Generic Localized Algorithm

Generic localized algorithm five components:

(i) Data acquisition mechanism

(ii) Optimization mechanism,

(iii) Search expansion rules

(iv) Bounding conditions

(v) Termination rules.

The data acquisition mechanism facilitates which sensed data is obtained from which node. The optimization mechanism provides a partial or complete solution to the targeted task. Search expansion rules indicate which nodes are best to contact next. Bounding conditions indicate which nodes should not be considered further, since information that they have is irrelevant for the final solution. Finally, termination criteria indicate when search expansion and optimization mechanism can be halted.

The idea is to request and process data only locally and only from nodes who are likely to contribute to both final solution as well as to provide good bounds to determine non-promising search directions.

3.2. Localization protocol

The localization protocols are divided into two categories: range-based and range-free. The former is defined by protocols that use absolute point-to-point distance estimates (range) or angle estimates for calculating location. The latter makes no assumption about the availability or validity of such information. Because of the hardware limitations of WASN devices, solutions in range-free localization are being pursued as a cost-effective alternative to more expensive range-based approaches.

3.2.1. Range-Based Localization Schemes

Time of Arrival (ToA) technology is commonly used as a means of obtaining range information via signal propagation time. The most basic localization system to use ToA techniques is GPS. GPS systems require expensive and energy-consuming electronics to precisely synchronize with a satellite's clock. With hardware limitations and the inherent energy constraints of sensor network devices, GPS and other ToA technology present a costly solution for localization in wireless sensor networks.

Time difference of arrival (TDoA) technique is similar to the ToA method. The only difference is, in this approach, the system assumes the presence of simultaneously emitted signals from two beacons and finds the time difference between the two signals. While many infrastructure-based systems have been proposed that use TDOA, additional work such as AHLos has employed such technology in infrastructure-free sensor networks. Like ToA technology, TDoA also relies on extensive hardware that is expensive and energy consuming, making it less suitable for low-power sensor network devices.

Angle of arrival (AoA) techniques measure the angle between a number of beacons (more than three) and an object to determine the position of the object. The precision of the AoA techniques diminishes with increasing the distances between the unknown object and the beacons. Received signal strength indicator (RSSI) techniques works by observing the power of the received signal. Assuming that the original power of the received signal at a transmitter is known, the propagation loss can be used to estimate the distance between the transmitter and the receiver. The system should have a map between the loss and the distances, which is often either an empirical table or an equation-based model. The errors in this type of distance measurement can be quiet high, due to the obstacles and the multi-path effects of the environment on the signal. This technique is mostly implemented with the radio frequency (RF) transmitter and receivers, and has been used with the GSM technology.

3.2.2. Range-Free Localization Schemes

In sensor networks and other distributed systems, errors can often be masked through fault tolerance, redundancy, aggregation, or by other means. Depending on the behavior and requirements of protocols using location information, varying granularities of error may be appropriate from system to system. Acknowledging that the cost of hardware required by range-based solutions may be inappropriate in relation to the required location precision, researchers have sought alternate range-free solutions to the localization problem in sensor networks.

A heterogeneous network containing powerful nodes with established location information is considered. In this work, anchors beacon their position to neighbors that keep an account of all received beacons. Using this proximity information, a simple centroid model is applied to estimate the listening nodes' location. Instead of single hop broadcasts, anchors flood their location throughout the network maintaining a running hop-count at each node along the way. Nodes calculate their position based on the received anchor locations, the hop-count from the corresponding anchor, and the average-distance per hop, a value obtained through anchor communication.

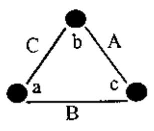
3.3. Localization techniques

There are three major classes of localization algorithms. These algorithms operate on an ad-hoc network of sensor nodes where a small percentage of the nodes are aware of their positions either through manual configuration or using GPS. Though these techniques are mainly associated to range-based techniques, they also find their use with range-free techniques too.

Once we measure the distances between an object and a number of beacons, we also need a way to combine the measurements to find the actual position. The most common methods to combine the distance measurements from three or more beacons are triangulation, simple trilateration (multilateration), and atomic multilateration.

3.3.1. Triangulation

Triangulation is a method for finding the position of a node, when the angles are measured by AoA technique. An example of such a procedure is shown in Figure 3.1. The object X measures its angles with respect to the beacons A_1, A_2 and A_3. The measured angles form three straight lines along the directions XA_1, XA_2 and XA_3. The intersection between the three lines defines the location of the node X. The accuracy of this technique is heavily dependent upon the accuracy of the employed angle measurement technique.



Sines rule $\frac{A}{\text{Sin a}} = \frac{B}{\text{Sin b}} = \frac{C}{\text{Sin c}}$

Cosine rule
$C^2 = A^2 + B^2 + 2AB \cos (c)$
$B^2 = A^2 + C^2 - 2BC \cos (b)$
$A^2 = B^2 + C^2 - 2BC \cos (a)$

Fig.3.1.Triangulation

3.3.2. Simple Trilateration

Simple trilateration is used when we have an accurate estimate of distances between a node and at least three beacon nodes. This simple method finds the intersection of three circles centered at beacons as the position of the node. The scenario is shown in Figure 3.2.

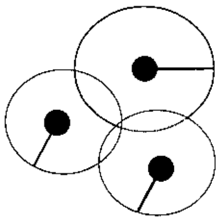


Fig.3.2.Trilateration

3.3.3. Multilateration

Atomic multilateration is accepted as the most appropriate way to determine the location of a sensor node based on locations of beacons. An example is shown in Figure 3.3., where the nodes A_1, A_2, A_3, and A_4 are beacons, with known estimates of their locations, while the node X estimates its location using a multilateration procedure. The procedure attempts to estimate the position of a node by minimizing the error and discrepancies between the measured values.

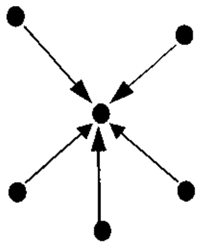


Fig.3.3. Multilateration

3.4. Proximity-based Localization Technique

Most of the proposed localization techniques today, depend on recursive trilateration/multilateration techniques. One way of considering sensor networks is taking the network to be organized as a hierarchy with the nodes in the upper level being more complex and already knowing their location through some technique (say, through GPS). These nodes then act as beacons by transmitting their position periodically. The nodes that have not yet inferred their position listen to broadcasts from these beacons and use the information from beacons with low message loss to calculate its own position. A simple technique would be to calculate its position as the centroid of all the locations it has obtained. This is called as proximity based localization.

It is quite possible that all nodes do not have access to the beacons. In this case, the nodes, which have obtained their position through proximity, based localization themselves act as beacons to the other nodes. This process is called iterative multilateration. As can be guessed, iterative multilateration leads to accumulation of localization error since the position estimated for the unknown node depends upon the previous estimations which themselves are not accurate values.

Thus the focus of this project is to analyze the impact of various design choices, especially the node density, beacon density and the range of each beacon on the accuracy and the effectiveness of the localization procedure. We describe accuracy in terms of the average localization error and effectiveness in terms of the number of the nodes that could resolve their position when the procedure convergences. (i.e., the point after which no further improvement is evidenced).

3.5. Localization error

Beacons situated at known positions (X_a, Y_a), transmit periodically with a time period T. Clients listen for a period T to evaluate connectivity. If the percentage of messages received from a beacon in a time interval t exceeds a threshold h, that beacon is considered connected. A client then estimates its position estimate (X_{est}, Y_{est}) as the centroid of the positions of all connected beacons. Given the actual position of the client, we can compute the localization error LE, which is the distance between the client's estimated and actual positions.

$$LE(X_a, Y_a) = [(X_{est} - X_a)^2 + (Y_{est} - Y_a)^2]^{1/2} \qquad (1)$$

3.6. Adaptive Beacon Placement

Given a localization algorithm, one must deploy a field of beacons as infrastructure, and then extend this field if it proves insufficient(The localization error is very high).

Beacon placement strongly affects the quality of spatial localization, a critical service for context-aware applications in wireless sensor networks; yet this aspect of localization has received little attention. Fixed beacon placement approaches such as uniform and very dense placement are not always viable and will be inadequate in very noisy environments in which sensor networks may be expected to operate (with high terrain and propagation uncertainties).

The approach to incremental improvement of localization through beacon placement is based on empirical adaptation. By adaptation, we mean we are improving the quality of localization by adjusting beacon placement or adding a few beacons rather than by completely re-deploying all beacons. By empirical, we mean the deployment of additional beacons is influenced by measurements of the operating localization system rather than by careful or complete off-line analysis of a complete system model. A simple choice is an off-line algorithm called HEAP-MAX with complete terrain exploration and no measurement noise.

3.7. HEAP-MAX Algorithm

The goal of this algorithm is to determine candidate points for placement of an additional beacon, so as to maximize the gains obtained. This is much better as compared to randomly selecting the candidate points and adding the beacons. The amount of complexity involved in such an algorithm is given by O[1].

The HEAP-MAX is much better than the random procedure and involves dividing the terrain into a number of squares. A predefined point is selected on the circumference of each square and the localization error is simulated at each of these points assuming that if there is a node at those points. Then the measured location error is sorted and a new beacon is added to the point corresponding to the highest localization error as shown in the fig.3.4.

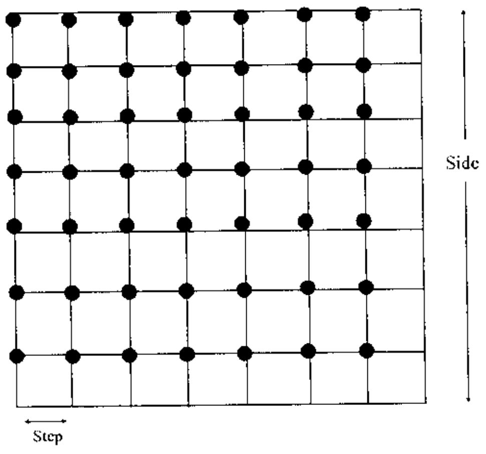


Fig.3.4. Heap-Max Illustration

This algorithm is predicated on the assumption that points with high localization error are spatially correlated. The advantage of this algorithm is that it can be computed in a very straightforward way.

3.8. Implementation Environment

C++ is an object oriented programming language. It was developed by Bjarne Stroustrup at AT&T Bell Laboratories in Murray Hill, New Jersey, USA, in th early 1980's. Therefore C++ is an extension of C with major additional class construct features of Simula67. Since the class was a major addition to original C language. Stroustrup initially called the new language as "C with Classes". However later name was changed as C++. The idea of C++ comes from the C increment operator ++. C++ is a superset of C. some rules that he uses for the design of C++ are :

¬ C++ is designed to be a statically typed, general-purpose language that is as efficient and portable as C

¬ C++ is designed to directly and comprehensively support multiple programming styles (procedural programming, data abstraction, object-oriented programming, and generic programming)

¬ C++ is designed to give the programmer choice, even if this makes it possible for the programmer to choose incorrectly

¬ C++ is designed to be as compatible with C as possible, therefore providing a smooth transition from C

¬ C++ avoids features that are platform specific or not general purpose

¬ C++ does not incur overhead for features that are not used C++ is designed to function without a sophisticated programming environment

Object-oriented Features

C++ introduces some object-oriented (OO) features to C. It offers classes, which provide the four features commonly present in OO (and some non-OO) languages: abstraction, encapsulation, inheritance and polymorphism.

Structures

A C++ structure is a datatype in the C++ programming language which, in addition to the properties of its C counterpart, can have its own member functions and operator overloads. It is identical to a C++ class except that its member accessibility defaults to public instead of private. As C++ structures are declared by the struct keyword, C++ structures are also known as structs, and many programmers use "struct" as a short form for "structure". C++ structures are declarated with the struct keyword. Members of a structure include variables (including other structs), functions (specific identifiers or overloaded operators), constructors and destructors.

Graphic Features

The graphics.h header file, graphics.lib library file and Graphics driver (BGI file) in the program folder are part of Turbo C++ package that enables graphics working in DOS shell itself. In graphics mode, all the screen co-ordinates are mentioned in terms of pixels. Number of pixels in the screen decides resolution of the screen.

To start the graphics system, one must first call initgraph. initgraph initializes the graphics system by loading a graphics driver from disk (or validating a registered driver) then putting the system into graphics mode.

The C++ graphics library supports a rich rest of functions like circle(), rectangle(),ellipse(),arc(),outtextxy() which enables the user to draw circles and other shapes.

Finally, the closegraph() function switches back the screen from graphics mode to text mode. It clears the screen also. A graphics program should have a closegraph function at the end of graphics. closegraph() is called after getch() since screen should not clear until user hits a key.

C++, with its enriched graphical features is ideal for simulating sensor networks. The nodes are represented using circles and the characteristics of the nodes like communication range, node status and its location estimate are put together in a structure. The flexibility of the language lets the simulation to be implemented in an interactive way.

RESULT ANALYSIS

4. ANALYSIS OF RESULTS OBTAINED (5)

4.1. Simulation Scenario

The simulations are performed in an 75 X 50 m rectangular area. Nodes are randomly placed over this region. The real locations of sensor nodes A_i, i=0,..,n are represented as points $Ai(x_i; y_i)$. Coordinates x_i and y_i are generated from two uniform distributions, one on the interval $[0,X_{max}]$ and one on the interval $[0,Y_{max}]$. For each simulation, a subset of nodes that have initial estimates of their locations is randomly or according to user specified criteria selected. The transmission range of each beacon is set to 20m. The prime goal is to estimate the positions of as many unknown nodes as possible in a fully localized fashion.

When the beacon placement is uniform, the centroid of the positions of all connected beacons is a feasible solution in the region of connectivity overlap. A client estimates its position (X_{est}, Y_{est}) to be the centroid of the positions of all connected beacon.

$$Wi = \frac{\left(1/r_i^{\ 2}\right)}{\left(\sum_{i=1}^{k} 1/r_i^{\ 2}\right)}$$

$$(X_{est}, Y_{est}) = \left(\sum_{i=1}^{k} W_i X_i, \sum_{i=1}^{k} W_i X_i \right)$$

and the estimated error is

$$LE(X_a, Y_a) = [(X_{est} - X_a)^2 + (Y_{est} - Y_a)^2]^{1/2}$$

4.2. Discussions on Trilateration

The goal of the study is to give an insight into the performance of the trilateration based localization procedure under different design choices. At the time of deployment of a sensor network to accomplish a given mission, there are a number of design parameters that are to be properly decided upon so as to get an optimum level of performance. There are many tunable parameters left to the choice of the network designer, like the node density, beacon density, beacon range etc (only w.r.t. to localization techniques, for brevity). These heuristics are chosen either based upon the application's optimality requirements or decided empirically by the network designer. In this study, we try to estimate the impact of the various design choices on the accuracy and efficacy of the localization procedure.

We generate 50 different topology and the results shown in graphs are the averages of the values measured in each simulation run. We start by considering the impact of beacon density on the accuracy of the localization procedure. We use the following measures for the evaluation of the algorithm:

(1) Beacon density (ρ)- denotes the number of beacons deployed per unit area

(2) BPRN (Beacons Per Radio Neighborhood)- denotes the number of beacons available per radio range of a node

$$\text{BPRN} = \rho.\Pi.\text{Range}^2$$

For instance, Fig. 3 shows the values of BPRN calculated for varying beacon densities.

(3) ARLE (Average Relative Localization Error)

This is the ratio of the average localization error of all nodes measured with n beacons to that with 3 beacons (a minimum number needed for trilateration).

ARLE is calculated using the following equation:

$$\left[\sum_{i=1}^{N} \sqrt{(Y_{i0}^{n}-Y_{i0})^2+(X_{i0}^{n}-X_{i0})^2/(Y_{i0}^{3}-Y_{i0})^2+(X_{i0}^{3}-X_{i0})^2}\right]/N$$

4.2.1. Beacon density Vs Efficiency (5)

Firstly, the impact of beacon density on the usefulness of the procedure is analysed. The graphs in Fig 4.1-4.3 illustrate how the beacon density influences the number of nodes resolved and the speed of convergence of the procedure. For this study, the node intensity is varied from 50 to 100.

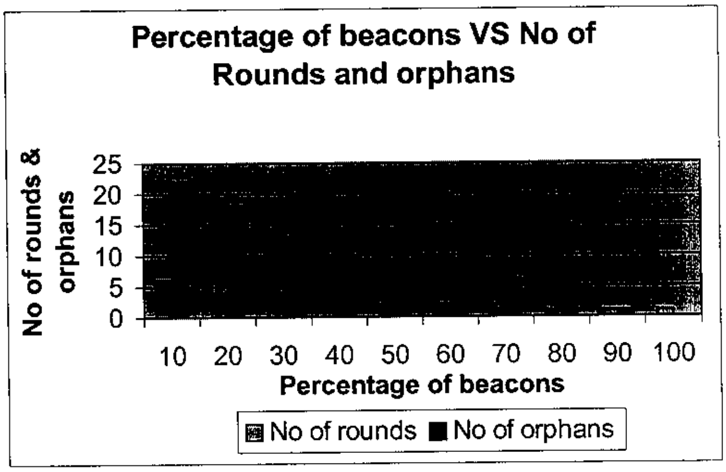


Fig 4.1. Number of nodes resolved for node intensity=50

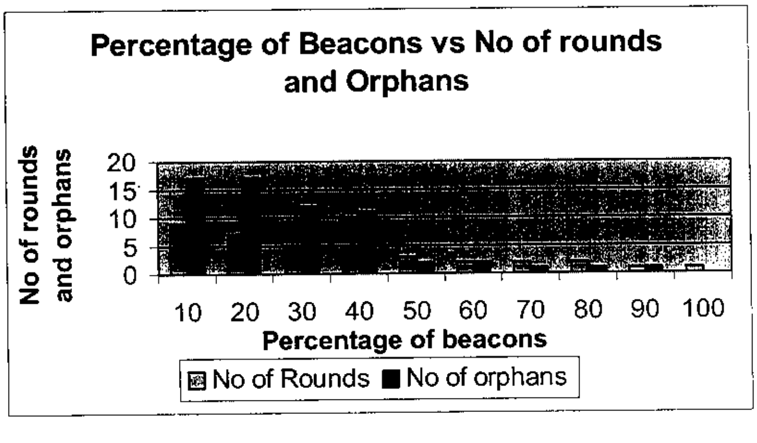


Fig 4.2. Number of nodes resolved for node intensity=75

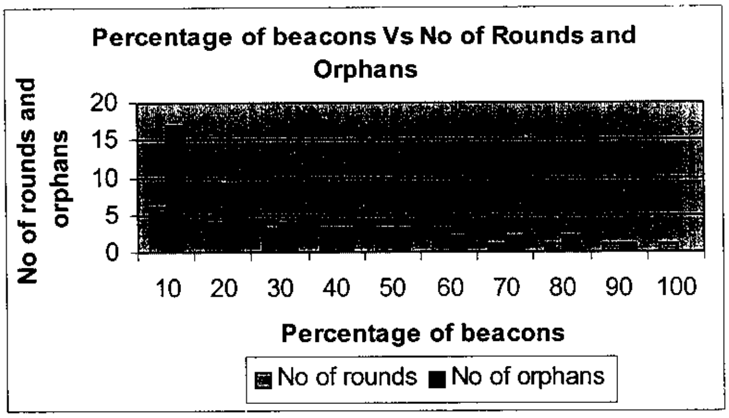


Fig 4.3 Number of nodes resolved for node intensity=100

As evident from these graphs, as the beacon density increases, the speed of the procedure increases and the number of orphans (nodes remaining unresolved of their location) decreases. Another more important observation is that, as the node intensity increases, the procedure converges very quickly because of the increased connectivity among the nodes

4.2.2. Beacon density Vs. Accuracy

We now proceed to analyze the impact of beacon density on the accuracy of the localization procedure, measured in terms of ARLE.

50 nodes

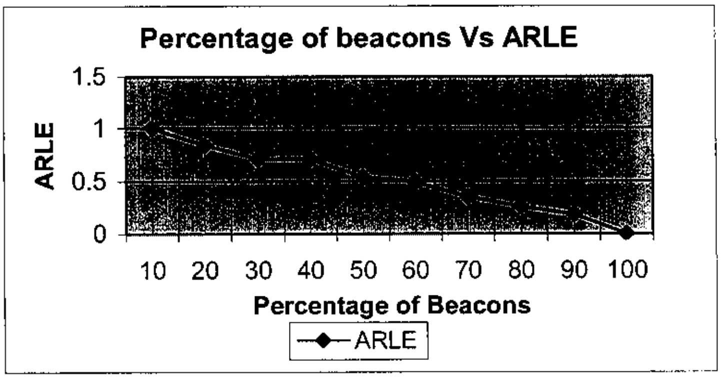


Fig 4.4 Localization Accuracy

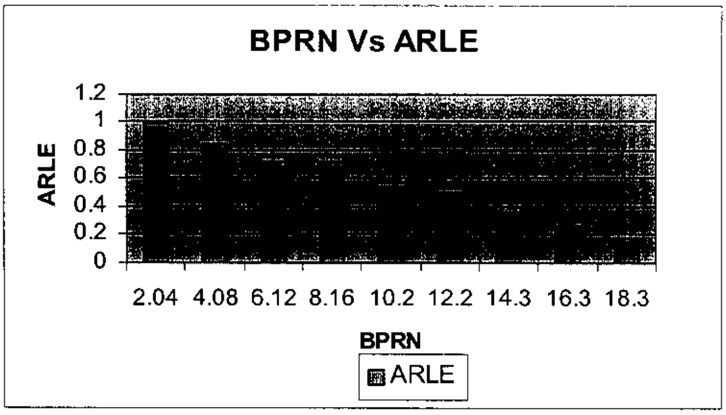


Fig. 4.5 Localization Accuracy w.r.t. BPRN

75 nodes

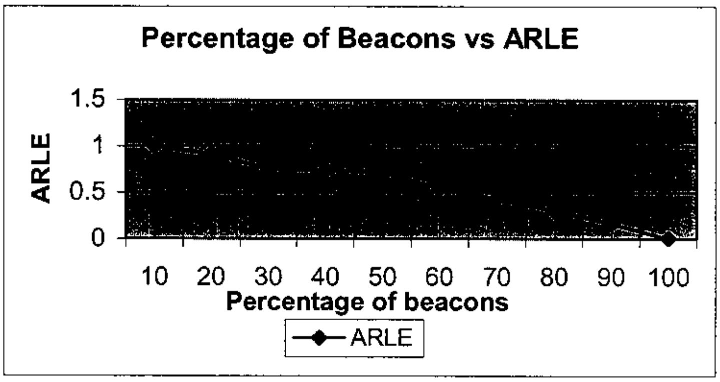


Fig 4.6 Localization Accuracy

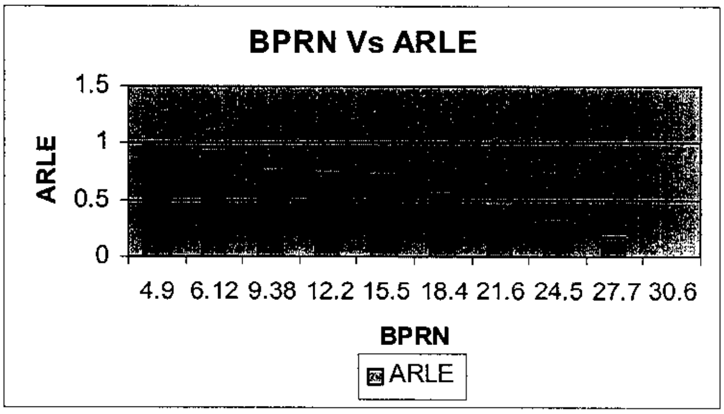


Fig. 4.7 Localization Accuracy w.r.t. BPRN

100 nodes

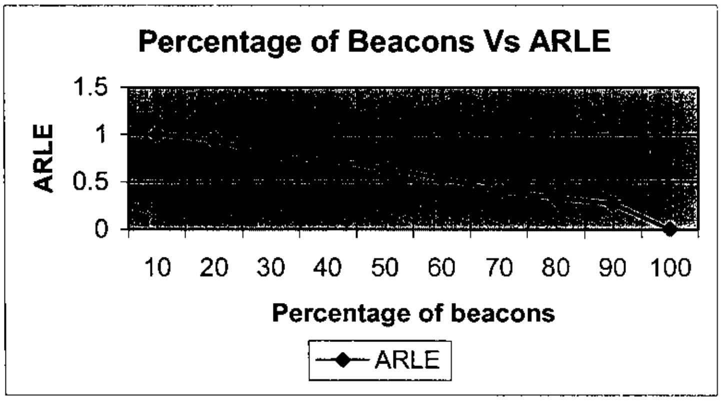


Fig 4.8 Localization Accuracy

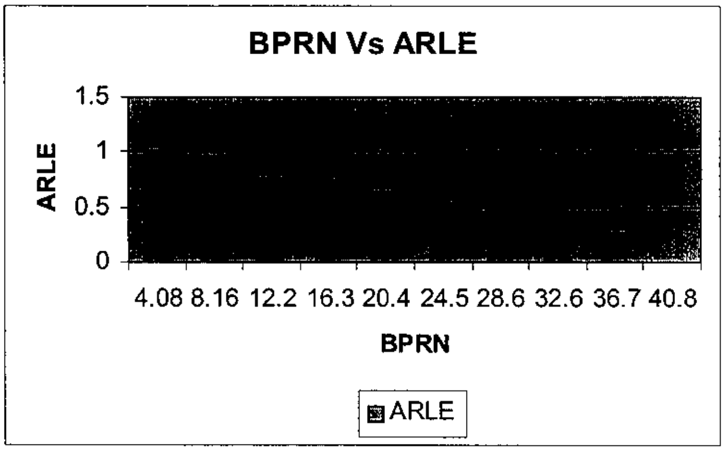


Fig. 4.9 Localization Accuracy w.r.t. BPRN

Figs.4.4-4.9 shows how the localization error decreases as more and more number of beacons are deployed. But practically, the trade off between the cost of the network and the accuracy of the locations discovered should be considered in deciding the optimum beacon density.

4.3. Discussions on HEAP-MAX

The result obtained using trilateration is further optimized by HEAP-MAX and the efficiency of the algorithm is checked by varying the node density between 50 - 100. The same parameters considered for trilateration is taken here also and the impact of various parameters on the beacon density are shown by the charts below.

4.3.1. Beacon density Vs Efficiency

The effect of beacon density on the speed of convergence and the number of orphans are presented for different node densities.

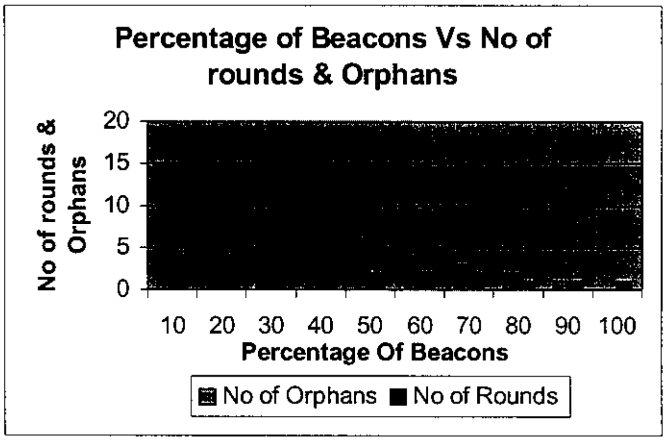


Fig 4.10. Number of nodes resolved for node intensity=50 (max)

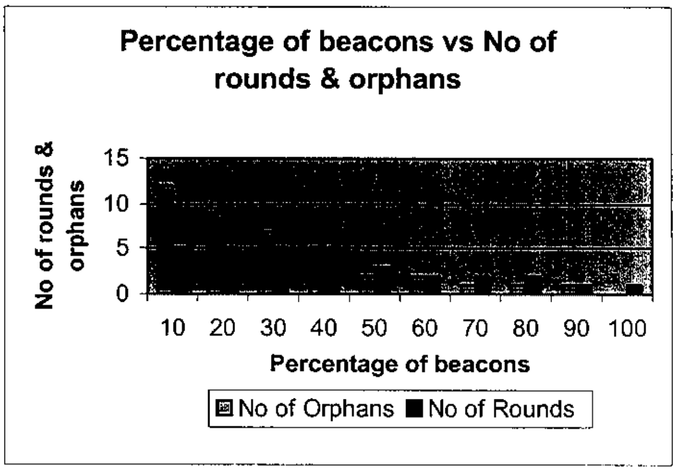


Fig 4.11. Number of nodes resolved for node intensity=75 (max)

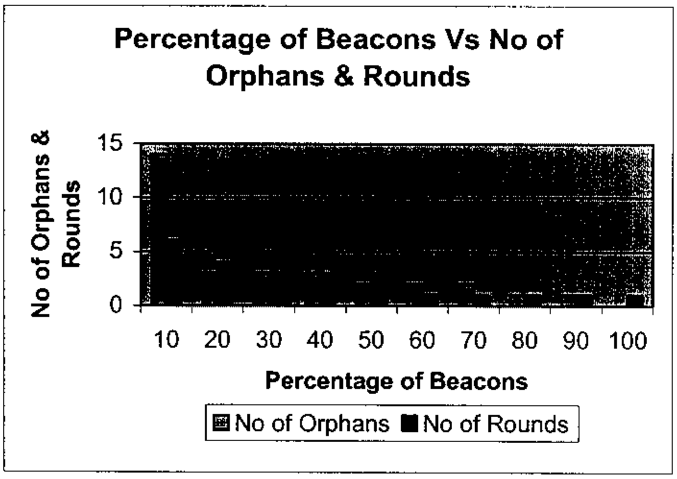


Fig 4.12. Number of nodes resolved for node intensity=100 (max)

4.3.2. Beacon density Vs. Accuracy

The effect of extra beacon deployment on the Average relative error and Beacon Per Radio Neighborhood are analysed and the results are shown by the graphs 4.13-4.18.

50 Nodes

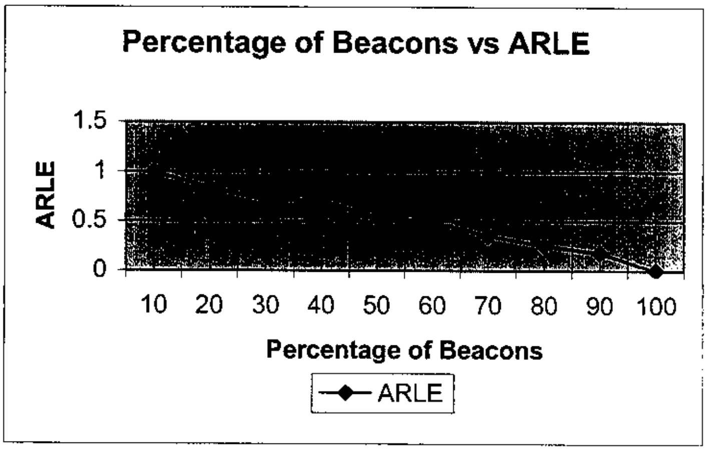


Fig 4.13 Localization Accuracy

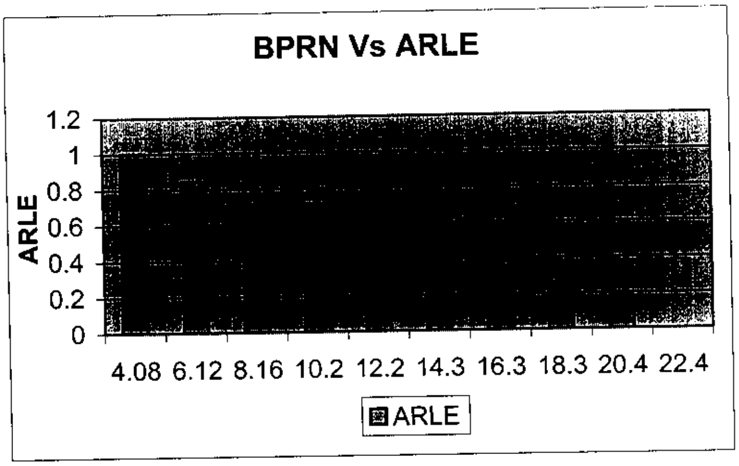


Fig. 4.14 Localization Accuracy w.r.t. BPRN

75 Nodes

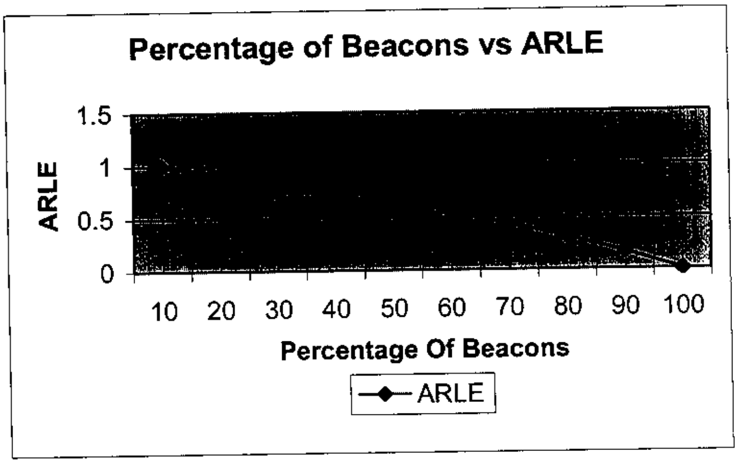


Fig 4.15. Localization Accuracy

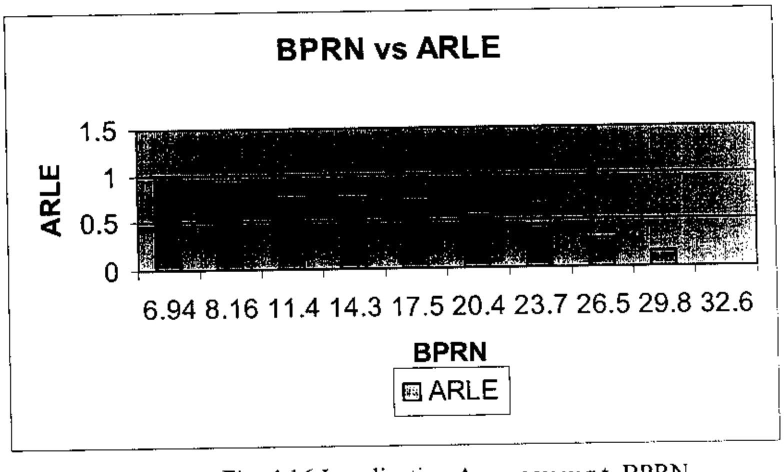


Fig. 4.16 Localization Accuracy w.r.t. BPRN

100 Nodes

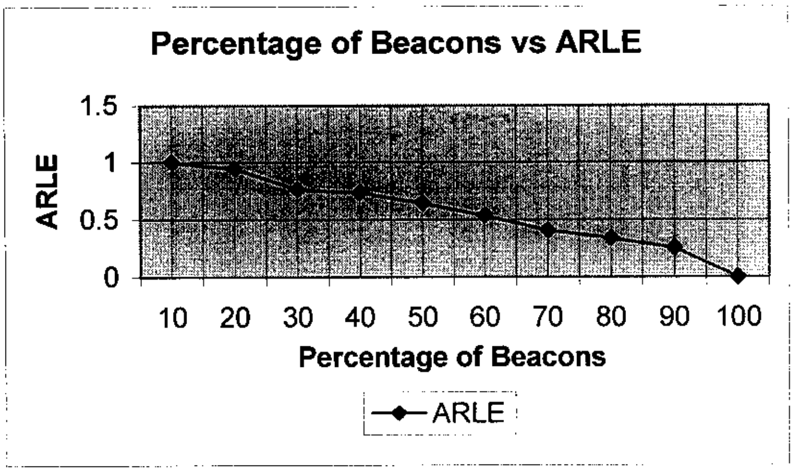


Fig 4.17. Localization Accuracy

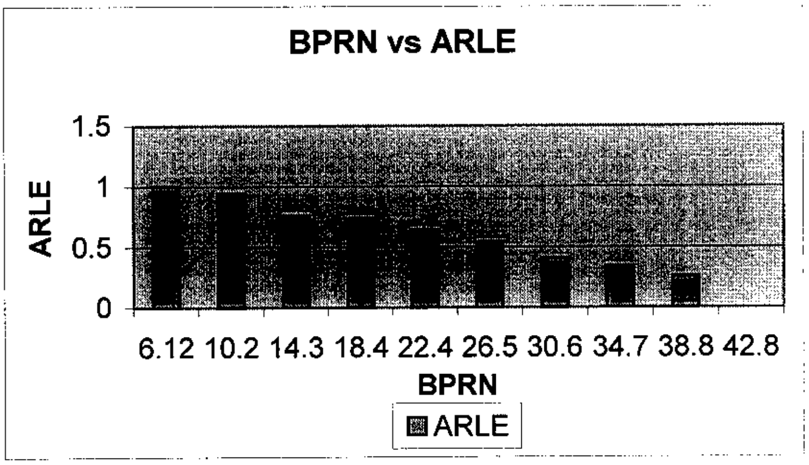


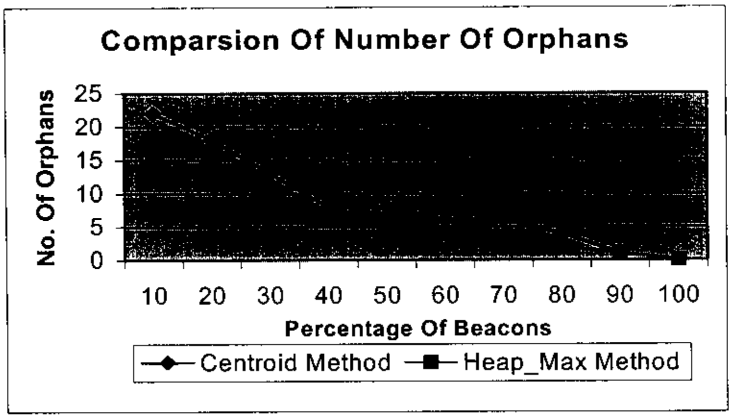
Fig. 4.18 Localization Accuracy w.r.t. BPRN

Thus it is evident from the above charts that the localization error reduces as the beacon density increases.

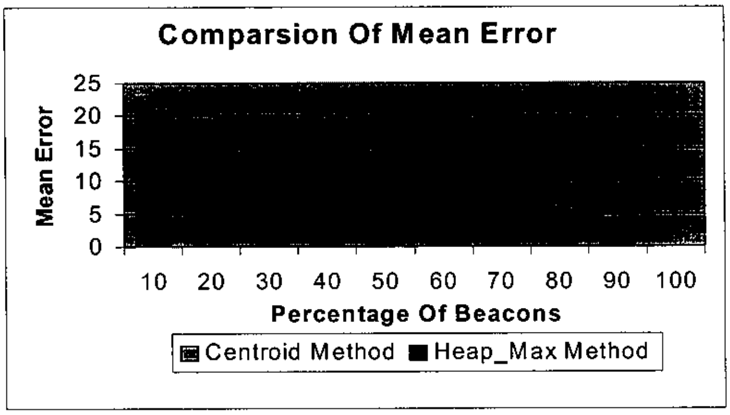
4.3. Comparison of performances Before and after optimization

Here the results obtained from the trilateration and the results obtained after Optimizing with max are discussed. The number of orphans, the number of rounds and the mean error are compared for different node setups like 50, 75 and 100. For every node setup the beacon density is varied from 10% to 100%. With the values obtained graphs are drawn.

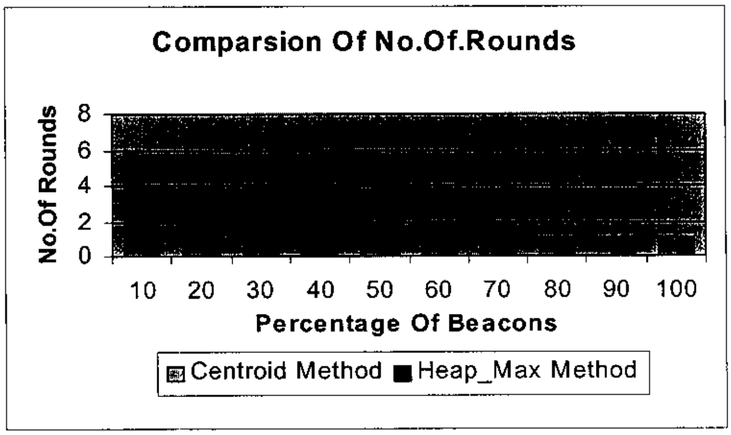
50 Nodes



4.19 Comparison w.r.t. Number of orphans

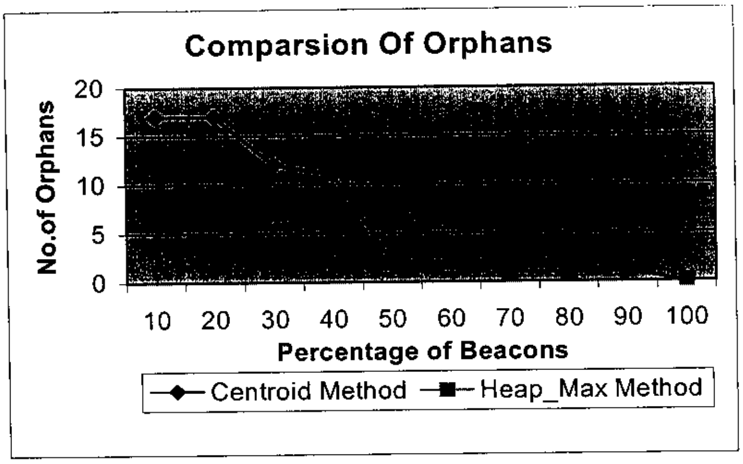


4.20 Comparison w.r.t. Mean Error

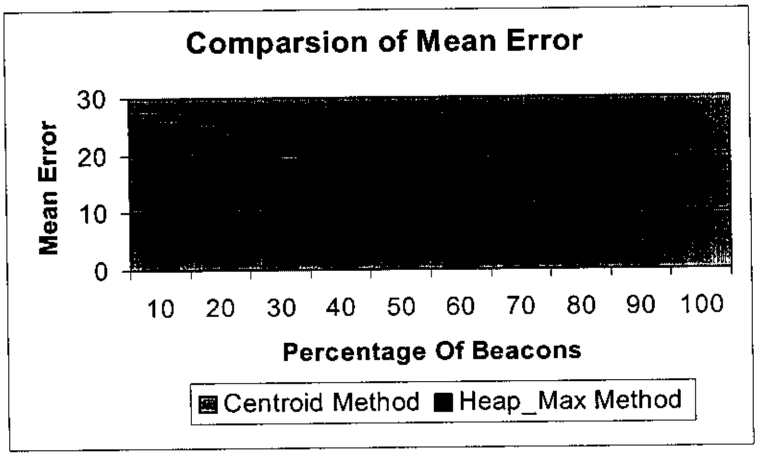


4.21 Comparison w.r.t. Number of Rounds

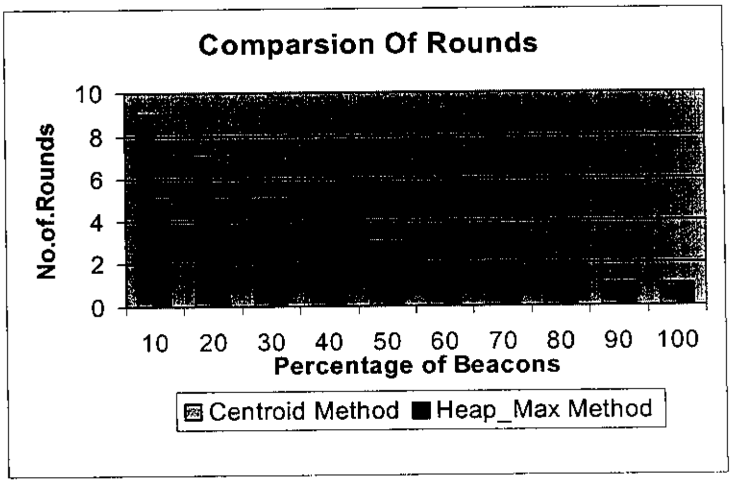
75 Nodes



4.22 Comparison w.r.t. Number of orphans

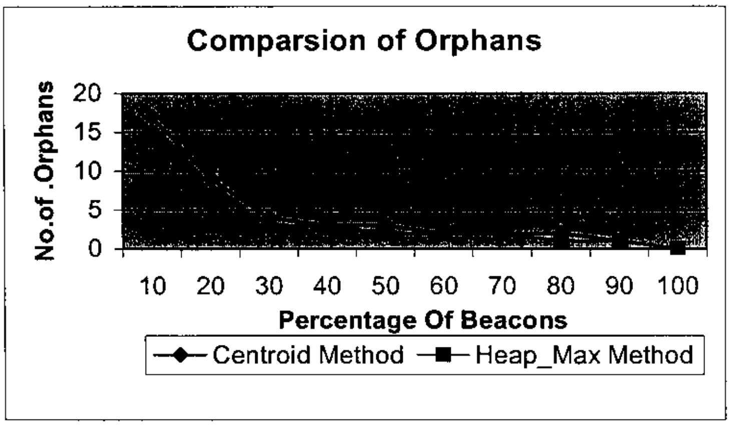


4.23 Comparison w.r.t Mean Error

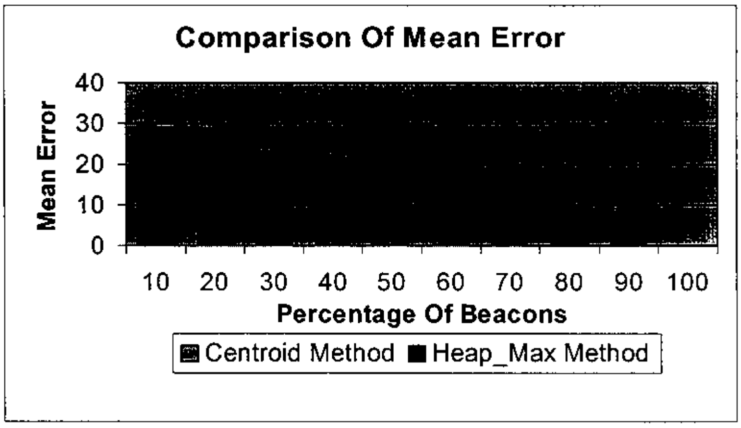


4.24 Comparison w.r.t Number of Rounds

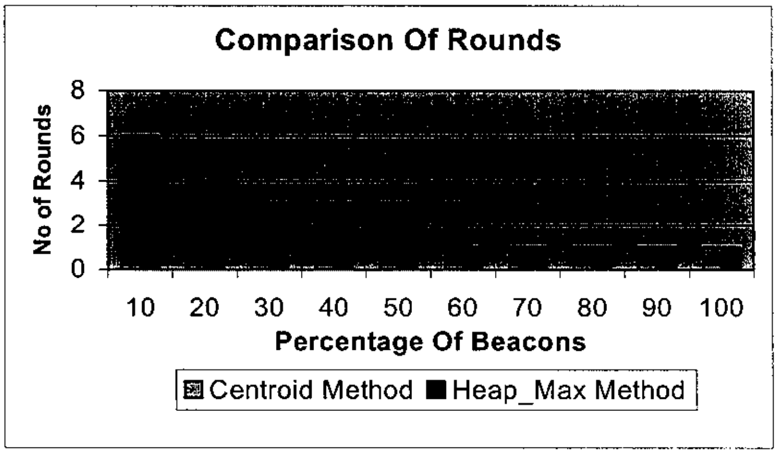
100 Nodes



4.25 Comparison w.r.t. Number of orphans



4.26 Comparison w.r.t Mean Error



4.27 Comparison w.r.t. Number of Rounds

Thus we see that the number of orphans, Mean error and the number of rounds is less for HEAP-MAX method when compared to centroid method because the additional beacons are deployed are deployed at optimal position,by carefully analyzing the field. After a certain period of time, when the beacons are increased, the values get saturated.

DESIGN DOCUMENTS

5. DESIGN DOCUMENTS

5.1. Introduction

A software design is a model of a real world system that has many participating entities and relationships. This design is used in a number of different ways. It acts as the basis for detailed implementation. It serves as a communication medium between the designers of the subsystems, it provides info to system maintainers about the original intentions of the system designers.

5.2. Input Design

The project needs the user to give appropriate inputs such as the number of nodes to be deployed and the number of beacons among the deployed nodes. If less than predefined values are given as input to the system, an error message is displayed to the user. Once, the input is obtained from the user, the specified node setup is generated by the system.

5.3. Output Design

Once the nodes are deployed as per the user requirements, trilateration is performed iteratively until there are no nodes that are left behind with three beacons. The optimization is done in-order to improve the quality of localization and the results are displayed to the user before and after the optimization.

5.4. Process Design

The process that is going to proceed will be based on the input provided to the system.

With the help of the input got from the nodes are deployed, the beacons are differentiated from the normal nodes by means of colour. The initialization phase starts by identifying the node status as GPS equipped or Beacons , Common, or orphans. A GPS equipped node is a beacon. A node is a common node if it is not a beacon and it has three beacons in the communication radius to trilaterate. If the node is neither GPS equipped nor it has three beacons in its range it is an orphan.

Once the node status is found out, the orphans and beacons are segregated. Trilateration is done for those common nodes that have three beacons in their range and the node status is changed as beacon. After iterating this location estimation several times , still there are nodes that cannot be trilaterated. They belong to the orphan community.

The Localization error is calculated by finding the difference between the actual position of the nodes and their estimated position. This localization error can be reduced by adaptive beacon placement (i.e. Adding a few other beacons at most optimal locations to improve the localization quality).

A simple algorithm called HEAP-MAX is used wherein the total landscape is divided into distinct squares and the location error is simulated at some point in each square. The additional beacons are deployed at points with highest measured localization error according to the degree of localization expected by the user.

PRODUCT TESTING

6. PRODUCT TESTING

Testing is done to detect the errors in the software. This implies not only to coding phase but to uncover errors introduced at all the previous phases.

6.1. Unit testing

Each and every module is tested separately to check if its intended functionality is met. Some unit testing performed are,

- Creation of nodes
- Beacon deployment
- Trilateration
- Optimization

6.2. Integration testing

It is the testing performed to detect errors on interconnection between modules. Here, the process of integrating location discovery and optimization procedure is performed and checked for proper execution. The connectivity to other modules is also tested.

6.3. System testing

The system is tested against the system requirements to see if all the requirements are met and if the system performs as per the specified requirements. The system is tested as a whole to check for it functionality.

6.4. Validation testing

The test is done to check for the validity of the entered input. The parameters are checked for improper entries. We also checked by providing wrong inputs which are not available.

FUTURE ENHANCEMENTS

7. FUTURE ENCHANCEMENTS

The beacon placement in the localized optimization is addressed in the context of proximity based localization approaches. An interesting point of comparison are beacon placement algorithms for multilateration based localization approaches, as the error characteristics of the two are significantly different. In the former approach, localization error is governed by beacon placement and density, whereas in the latter approach, it is influenced by the geometry of the beacon nodes. The existing beacon placement algorithms can be recast for multilateration based localization approaches.

CONCLUSION

8. CONCLUSION

Iterative multilateration procedures play an important role in location discovery problem in Ad-Hoc wireless sensor networks. Trilateration, a proximity based technique is one such example: given the inherent resource constraints of WASNs and the desired localization accuracy of the intended application, it is regarded as a cost effective and sufficient alternative for more expensive range based techniques.

The distinguished advantage of this localization scheme is its simplicity and ease of implementation. To verify the effectiveness of the procedure the analysis is done by considering the impact of one of the most important architectural parameter, viz. the beacon density on the accuracy. It is also found that optimization in all these parameters can be done using the HEAP-MAX. From this study, it is evident that, though this technique is coarse-grained leading to significant localization errors, may prove useful for certain non-critical applications.

APPENDICES

SAMPLE CODE

APPENDIX

A. Sample code

```
#include<iostream.h>
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include<math.h>
#include<dos.h>
#include<alloc.h>

struct point
{
        float x;
        float y;
}p,ep,b[15];

struct node
{
        struct point p,ep,b[15];
        int bc;
        int fl;
        int o;
        int range;
        int gps;
        float e;
}n[125];

//function declarations

void max_location (float[130],float[130],float[30][130],float[30][130],int[130],int[130],
                                                                        int,int[130]);
void max_trilaterate (float[30][130],float[30][130],float[130],float[130],float,float,
                                                                          float,int);
void location_info (struct node n[125]);
point trilaterate (point,point,point,float,float,float);
int bcount,ncnt,bcnt;
float ae;


int main (void)
{
        /* request auto detection of the graphic driver*/
        int gdriver = DETECT, gmode, errorcode;
int midx, midy;
int radius = 2;
//struct node n[125];
int b1[125],tmp[20];
int ocount = 0;
float te = 0.0;
char c,c1;
int bool[125];
int round = 0,flag = 0,bl = 0;
FILE *fi;

//for loop for initializing all the variables

for (int l1 = 0;l1 < ncnt;l1 + +)
{
        bool[l1]   = 0;
        n[l1].e    = 0.0;
        n[l1].p.x  = 0.0;
        n[l1].p.y  = 0.0;
        n[l1].ep.x = 0.0;
        n[l1].ep.y = 0.0;
}
/* initialize graphics and local variables */
initgraph (&gdriver, &gmode, "");


/* read result of initialization */
errorcode = graphresult ();
if (errorcode != grOk)  /* an error occurred */
{
        printf ("Graphics error: %s\n", grapherrormsg(errorcode));
        printf ("Press any key to halt:");
        getch ();
        exit (1); /* terminate with an error code */
}


//for loop to draw lines for good display
g2:
for (int s = 15;s < 700;s + +)
{
        putpixel (s,40,4);
        putpixel (s,50,14);
        putpixel (s,430,24);
        putpixel (s,440,34);
        delay (7);
```

```
          putpixel (40,s,44);
          putpixel (50,s,54);
          putpixel (550,s,55);
          putpixel (560,s,56);
}

// input and output streams used to get the no of nodes and beacons

gotoxy (25,15);
cout<<"Enter the Number of Nodes: ";
cin>>ncnt;
gotoxy (25,17);
cout<<"Enter the number of beacons: ";
cin>>bcnt;
if (bcnt < 3)
{
          gotoxy (25,19);
          cout<<"Beacons should be greater than 3";
          for (int h = 0;h < 640;h + +)
          {
                    for (int l = 0;l < 480;l+ +)
                    {
                              putpixel (h,l,0);
                    }
          }
          goto g2;
}

//for loop used to clear the screen

for (int h = 0;h < 640;h + +)
{
          for (int l = 0;l < 480;l + +)
          {
                    putpixel (h,l,0);
          }
}

//for loop used to randomly generate and draw nodes

// randomize ();
for (int i = 0;i < ncnt;i + +)
{
          midx = random (640);
          n[i].p.x = midx;
          //cout<<midx;
```

```
        //cout<<"\t";
        midy = random (480);
        n[i].p.y = midy;
        //cout<<midy;
        //cout<<"\n";
        setcolor (7);
        putpixel (n[i].p.x,n[i].p.y,7);
        /* draw the circle */
        circle (midx, midy, radius);
        /*if(ncnt<50)
        n[i].range = 300;
        else*/
        n[i].range = 100;
}
/*for loop used to randomly select beacons among the nodes drawn change their color
to indicate that they are beacons*/

for (int j = 0;j < bcnt;j + +)
{
        flag = 0;
        int r = random (ncnt);
        b1[j] = r;
        for (int h = 0;h < j;h + +)
        {
                if (r = = b1[h])
                {
                        j--;
                        flag = 1;
                        break;
                }
        }

        if (flag = = 0)
        {
                getpixel (n[r].p.x,n[r].p.y);
                putpixe l(n[r].p.x,n[r].p.y,6);
                setcolor (4);
                circle (n[r].p.x,n[r].p.y,2);
                n[r].range = 200;
                n[r].fl = 1;
                n[r].gps = 1;
                n[r].ep.x = n[r].p.x;
                n[r].ep.y = n[r].p.y;
        }
        // circle (a[r].x,a[r].y,200);
}
```

```
//initialization phase


location_info (n);

/*repeatedly look for the nodes that can estimate their location by
  trilateration and perform trilateration by calling location_info()
  and trilateration()*/

while (bl = = 0)
{
        round + +;
        // location_info ();
        bl=1;
        for (int k = 0;k < ncnt;k + +)
        {
                //int dist[15];
                if ((n[k].fl = = 0)&&(n[k].o = = 0))
                {
                        float d1 = sqrt (fabs(((n[k].p.x-b[0].x)*(n[k].p.x-b[0].x))-
                                            ((n[k].p.b[0].y)*(n[k].p.y- b[0].y))));
                        float d2 = sqrt (fabs(((n[k].p.x-b[1].x)*(n[k].p.x-b[1].x))-
                                            ((n[k].p.y-[1].y)*  (n[k].p.y-1].y))));
                        float d3 = sqrt (fabs(((n[k].p.x-b[2].x)*(n[k].p.x-b[2].x))-
                                             ((n[k].p.y-[2].y)*(n[k].p.y- b[2].y))));
                        n[k].ep = trilaterate (n[k].b[0],n[k].b[1],n[k].b[2],d1,d2,d3);
                        getpixel (n[k].p.x,n[k].p.y);
                        putpixel (n[k].p.x,n[k].p.y,6);
                        setcolor (4);
                        circle (n[k].p.x,n[k].p.y,2);
                        n[k].fl = 1;
                        n[k].o = 0;
                }
        }
        location_info (n);

        /*for loop used to check whether all the possible nodes can
            estimate their locations*/


        for (int y = 0;y < ncnt;y + +)
        {
                if (n[y].fl = = 1||n[y].o = = 1)
                bool[y] = 1;
                else

                {
                        bool[y] = 0;
                        bl = 0;
                }
        }

}
/* for (int g = 25;g < 50;g + +)
{  //for (int m = 0;m < 480;m + +)
//{
        cout<<n[g].fl;
        cout<<"\t";
        cout<<n[g].o;
        cout<<"\n";
        //clrscr ();
        //cout<<getpixel(n[g].p.x,n[g].p.y);
// }

}*/

//for loop used to count the number of orphans

for (int h1 = 0;h1 < ncnt;h1 + +)
{
        if (n[h1].o = = 1)
        {
                ocount + +;
                continue;
        }
        else
        {
                n[h1].e = sqrt(fabs(((n[h1].ep.x-n[h1].p.x)*(n[h1].ep.x-n[h1].p.x))-
                                ((n[h1].ep.y-n[h1].p.y)*(n[h1].ep.y-n[h1].p.y))));
                te = te + n[h1].e;
        }
}
ae = te / ncnt;

//for loop used to clear the screen

for (h = 0;h < 640;h + +)
{
        for (int l = 0;l < 480;l + +)
```

```
            {
                    putpixel (h,l,0);
            }
    }

    //for loop used to draw lines for good display

  for (s = 15;s < 700;s + +)
  {
            putpixel (s,40,4);
            putpixel (s,50,14);
            putpixel (s,430,24);
            putpixel (s,440,34);
            delay (7);
            putpixel (40,s,44);
            putpixel (50,s,54);
            putpixel (550,s,55);
            putpixel (560,s,56);
  }

  //output statements to display the result

  gotoxy (25,13);
  cout<<"The No of Orphans: ";
  cout<<ocount;
  cout<<"\n";
  gotoxy (25,15);
  cout<<"The Mean Error: ";
  cout<<ae;
  cout<<"\n";
  gotoxy (25,17);
  cout<<"The No of Rounds to terminate: ";
  cout<<round;
  cout<<"\n";
  gotoxy (25,19);
  cout<<"Run Heap max algorithm y/n: ";

  s1:

  while (!kbhit);
  c1 = getch();
  //cin>>c1;

  //max algorithm
  if (c1 = = 'y')
```

```
{
        //variable initialization

        //int step = 50;
        int bx[125],by[125],mfl[130],mor[130],mbcount[130];
        float mx[130],my[130],mxb[30][130],myb[30][130];
        float mxe[130],mye[130];
        double me[130],mte=0.0,mae;
        int m = 0,mround = 0,ml = 0;
        //int flag2 = 0;
        int mflag1[130];
        //float *mxp,*myp;
        //mx = &mx[0];
        //my = &my[0];

        //for loop used to clear the screen
        for (int kl = 0;kl < 640;kl + +)
        {
                for (int pl = 0;pl < 480;pl + +)
                {
                        putpixel (kl,pl,0);
                }
        }
        //for loop used to initialize variables

        for (int lp = 0;lp < 130;lp + +)
        {
                mx[lp] = 0.0;
                my[lp] = 0.0;
                for (int lp1 = 0;lp1 < 30;lp1 + +)
                {
                        mxb[lp1][lp] = 0.0;
                        myb[lp1][lp] = 0.0;
                }
        }

        //for loop used to draw the beacons

        for (int l = 0;l < ncnt;l + +)
        {
                if (n[l].gps = =1)
                {
                        bx[l] = n[l].p.x;
                        by[l] = n[l].p.y;
                        getpixel (n[l].p.x,n[l].p.y);
                        putpixel (n[l].p.x,n[l].p.y,6);
                setcolor (4);
                circle (n[l].p.x,n[l].p.y,2);
                //cout<<n[l].p.x<<"\t"<<n[l].p.y<<"\n";
        }
        /*else
        {
                putpixel (n[l].p.x,n[l].p.y,0);
                setcolor (0);
                circle (n[l].p.x,n[l].p.y,2);
        }*/
}

//for loop used to draw the points of square (top left corner)

for (int i1 = 1,i2 = 0;i1 <= 32 && i2 <= 640;i1+ +,i2 = i2 + 50)
{
        for (int j1 = 1,j2 = 0;j1 <= 24 && j2 <= 480;j1+ +,j2 = j2 + 50)
        {
                //rectangle (i,j,i +step,j + step);

                int a = i2;
                int b = j2;
                /*for (int t = 0;t < 5;t + +)
                {
                        if (bx[t] = = a && by[t] = = b)
                        {
                                flag2 = 1;
                        }
                }*/
                //circle (a,b,1);
                //if (flag2 = = 0)
                //{
                        getpixel (a,b);
                        putpixel (a,b,7);
                        mx[m] = a;
                        my[m] = b;
                        m + +;
                //}
        }
}

/*for (int p = 121;p < 130;p + +)
{
        cout<<mx[p]<<"\t"<<my[p]<<"\n";
}*/
```

```
for (int mfl1 = 0;mfl1 < m;mfl1 + +)
{
        mflag1[mfl1] = 0;
        mxe[mfl1] = 0.0;
        mye[mfl1] = 0.0;
        me[mfl1] = 0.0;
}

max_location (mx,my,mxb,myb,mfl,mor,m,mbcount);
while (ml = = 0)
{
        mround + +;
        ml = 1;

        for (int mp = 0;mp < m;mp + +)
        {
                float dist2[130];
                if (mfl[mp] = = 0 && mor[mp] = = 0)
                {

                        for (int tl2 = 0;tl2 < mbcount[mp];tl2 + +)
                        {
                                dist2[tl2]        =        sqrt        (fabs(((mx[mp]-
                                mxb[tl2][mp])*(mx[mp]-mxb[tl2][mp]))-
                                ((my[mp]-myb[tl2][mp])*(my[mp]-
                                myb[tl2][mp]))));
                        }
                        int tfl2 = 0;
                        float temp1 = 0.0,temp2 = 0.0;
                        for (tl2 = 0;tl2 < mbcount[mp]-1;tl2 + +)
                        {
                                for (int ty2 = 0;ty2<mbcount[mp]-tl2;ty2++)
                                {
                                        if (dist2[ty2] < dist2[ty2+1])
                                        {
                                           temp1 = mxb[ty2][mp];
                                           temp2 = myb[ty2][mp];
                                           mxb[ty2][mp] = mxb[ty2+1][mp];
                                           myb[ty2][mp] = myb[ty2+1][mp];
                                           mxb[ty2+1][mp] = temp1;
                                           myb[ty2+1][mp] = temp2;
                                           tfl2 = 1;
                                        }
                                }
                                if (tfl2 = = 0)
```

```
                                        break;
                                }
                                float d1= sqrt(fabs(((mx[mp]-mxb[0][mp])*
        (mx[mp]-mxb[0][mp]))-((my[mp]- myb[0][mp])*(my[mp]-myb[0][mp]))));
        float d2 = sqrt(fabs(((mx[mp]-mxb[1][mp])*
        (mx[mp]-mxb[1][mp]))-((my[mp]-myb[1][mp]) * (my[mp]-myb[1][mp]))));
                                float d3=sqrt(fabs(((mx[mp]-mxb[2][mp]) *     (mx[mp]-
mxb[2][mp]))-((my[mp]-myb[2][mp])*(my[mp]-myb[2][mp]))));
                                max_trilaterate (mxb,myb,mxe,mye,d1,d2,d3,mp);
                                mfl[mp] = 1;
                                mor[mp] = 0;
                                getpixel (mx[mp],my[mp]);
                                putpixe l(mx[mp],my[mp],6);
                        }
                }
                max_location (mx,my,mxb,myb,mfl,mor,m,mbcount);
                for (int mo = 0;mo < m;mo + +)
                {
                        if (mfl[mo] = = 1||mor[mo] = = 1)
                        {
                                mflag1[mo] = 1;
                        }
                        else
                        {
                                mflag1 [mo] = 0;
                                ml = 0;
                        }
                }
        }
        //cout<<mround;
        for (int p = 0;p < m;p + +)
        {
                //cout<<mxe[p]<<"\t"<<mye[p]<<"\n";
                me[p] = sqrt (fabs(((mxe[p]-mx[p])*(mxe[p]-mx[p]))-
                                ((mye[p]-my[p])*(mye[p]-my[p]))));
                mte = mte + me[p];
        }
        //mae = mte /135;
        //cout<<mae;

        /* for (int ep = 21;ep < 40;ep + +)
        {
                cout<<mx[ep]<<"\t"<<my[ep]<<"\t"
                <<mxe[ep]<<"\t"<<mye[ep]<<"\t"<<me[ep]<<"\n";
        }*/
        double me1[130];
```

```
for (int tl3 = 0;tl3 < m;tl3 + +)
{
        me1[tl3] = me[tl3];
}
int tfl3 = 0;
double temp = 0.0;
float temp1 = 0.0;
float temp2 = 0.0;
for (tl3 = 0;tl3 < m;tl3 + +)
{
        for (int ty3 = 0;ty3 < m-tl3;ty3 + +)
        {
                if (me[ty3] < me[ty3+1])
                {
                        temp = me[ty3];
                        temp1 = mx[ty3];
                        temp2 = my[ty3];
                        me[ty3] = me[ty3+1];
                        mx[ty3] = mx[ty3+1];
                        my[ty3] = my[ty3+1];
                        me[ty3+1] = temp;
                        mx[ty3+1] = temp1;
                        my[ty3+1] = temp2;
                        tfl3 = 1;
                }
        }
        if (tfl3 = = 0)
        break;
}
for (kl = 0;kl < 640;kl + +)
{
        for (int p = 0;pl < 480;pl + +)
        {
                putpixel (kl,pl,0);
        }
}
for (l = 0;l < ncnt; l + +)
{
        if (n[l].gps = =1)
        {
                bx[l] = n[l].p.x;
                by[l] = n[l].p.y;
                getpixel (n[l].p.x,n[l].p.y);
                putpixel (n[l].p.x,n[l].p.y,6);
                setcolor (4);
                circle (n[l].p.x,n[l].p.y,2);
        }
        else
        {
                getpixel (n[l].p.x,n[l].p.y);
                putpixel (n[l].p.x,n[l].p.y,7);
                setcolor (7);
                circle (n[l].p.x,n[l].p.y,2);
        }

}
for (int jk = 129;jk > 124;jk--)
{
        getpixel (mx[jk],my[jk]);
        putpixel (mx[jk],my[jk],6);
        setcolor (2);
        circle (mx[jk],my[jk],2);
}
int bool[125];
for (l1 = 0;l1 < ncnt;l1 + +)
{
        bool[l1] = 0;
        n[l1].e = 0.0;
        n[l1].ep.x = 0.0;
        n[l1].ep.y = 0.0;
}
int round = 0,bl = 0,ocount = 0;
location_info (n);
while (bl = =0)
{
        round + +;
        // location_info ();
        bl = 1;
        for (int k = 0;k < ncnt;k + +)
        {
                int dist[15];
                if ((n[k].fl = = 0)&&(n[k].o = = 0))
                {
                        for (int tl = 0;tl < n[k].bc;t l+ +)
                        {
                                dist[tl] = sqrt(fabs(((n[k].p.x-b[tl].x)*
                (n[k].p.x-b[tl].x))-((n[k].p.y- b[tl].y)*(n[k].p.y-b[tl].y))));
                        }
                int tfl = 0;
                struct point temp;
                temp.x = 0.0;
                temp.y = 0.0;
```

```
                    for (tl = 0;tl < n[k].bc-1;tl + +)
                    {
                            for(int ty = 0;ty < n[k].bc-tl;ty + +)
                            {
                                    if (dist[ty] > dist[ty+1])
                                    {
                                            temp = b[ty];
                                            b[ty] = b[ty+1];
                                            b[ty+1] = temp;
                                            tfl = 1;
                                    }
                            }
                            if (tfl = = 0)
                            break;
                    }
                    float d1 = sqrt(fabs(((n[k].p.x-b[0].x)*
            (n[k].p.x-b[0].x))-((n[k].p.y-b[0].y)*(n[k].p.y-b[0].y))));
                    float d2 = sqrt(fabs(((n[k].p.x-b[1].x)*
            (n[k].p.x-b[1].x))-((n[k].p.y-b[1].y)*(n[k].p.y-b[1].y))));
                    float d3 = sqrt(fabs(((n[k].p.x-b[2].x)*
            (n[k].p.x-b[2].x))-((n[k].p.y-b[2].y)*(n[k].p.y-b[2].y))));
                    n[k].ep = trilaterate(n[k].b[0],n[k].b[1],n[k].b[2],d1,d2,d3);
                    getpixel (n[k].p.x,n[k].p.y);
                    putpixel (n[k].p.x,n[k].p.y,6);
                    setcolor (4);
                    circle (n[k].p.x,n[k].p.y,2);
                    n[k].fl = 1;
                    n[k].o = 0;
            }
    }
    location_info (n);
```

/*for loop used to check whether all the possible nodes hav estimated their locations except those which lack 3 beacons*/

```
    for (int y = 0;y < ncnt;y+ +)
    {
            if (n[y].fl = = 1||n[y].o = = 1)
            bool[y] = 1;
            else
            {
                    bool[y] = 0;
                    bl = 0;
            }
    }
```

```
}

//forloop used to calculate the number of orphans

for(h1 = 0;h1 < ncnt;h1 + +)
{
        if(n[h1].o = = 1)
        {
                ocount + +;
                continue;
        }
        else
        {
                n[h1].e = sqrt (fabs(((n[h1].ep.x-n[h1].p.x)*
        (n[h1].ep.x-n[h1].p.x))-((n[h1].ep.y-n[h1].p.y)*(n[h1].ep.y-n[h1].p.y))));
                te = te + n[h1].e;
        }
}
ae = te / (ncnt+5);

// for loop used to clear the screen

for (kl = 0;kl < 640;kl + +)
{
        for(int pl = 0;pl < 480;pl + +)
        {
                putpixel (kl,pl,0);
        }
}

//for loop to draw lines for good display

for (s = 15;s < 700;s + +)
{
        putpixel (s,40,4);
        putpixel (s,50,14);
        putpixel (s,430,24);
        putpixel (s,440,34);
        delay (7);
        putpixel (40,s,44);
        putpixel (50,s,54);
        putpixel (550,s,55);
        putpixel (560,s,56);
}

// output statements used to display the result
```

```
        gotoxy (25,9);
        cout<<"After executing Heap Max";
        gotoxy (25,13);
        cout<<"The No of Orphans: ";
        cout<<ocount;
        cout<<"\n";
        gotoxy (25,15);
        cout<<"The Mean Error: ";
        cout<<ae;
        cout<<"\n";
        gotoxy (25,17);
        cout<<"The No of Rounds to terminate: ";
        cout<<round;
        }
        else if (c1 = = 'n')
        {
                exit (0);
        }
        else
        {
                goto s1;
        }
        getch ();
        closegraph ();
        return  0;
}

/* This is same as Loation_info() but it is used for gathering neighborhood
  information of the squarely deployed points where a node is assumed
  to be present*/


void max_location (float mx[130],float my[130],float mxb[30][130],float
                    myb[30][130],int mfl[130],int mor[130],int m,int mbcount[130])
{
        for(int k = 0;k < m;k + +)
        {
                int a = 0;
                if (getpixel (mx[k],my[k]) = = 6)
                {
                        mfl[k] = 1;
                        mor[k] = 0;
                        continue;
                }
                if (getpixel(mx[k],my[k]) = = 7)
        {
                for (int h = mx[k]-100;h <= mx[k]+100;h + +)
                {
                        if (h < 0||h > 640)
                        continue;
                        for (int g = my[k]-100;g <= my[k]+100;g + +)
                        {
                                if (g < 0||g > 480)
                                continue;
                                if (getpixel(h,g) = = 6)
                                {
                                        mxb[a][k] = h;
                                        myb[a][k] = g;
                                        a++;
                                        mbcount[k] = a;
                                }

                        }
                }
                if (a >= 3)
                {
                        mfl[k] = 0;
                        mor[k] = 0;
                }
                else
                {
                        mfl[k] = 0;
                        mor[k] = 1;
                }

        }
    }
}


/* This function is used to gather location information from the
  neighborhood of a node. Trilateration can be performed only
  when there are 3 nodes around a node that is willing to estimate
  its location.Beacons can be greater than 3 also*/

void location_info (struct node n[125])
{
        for (int k = 0;k < ncnt;k + +)
        {
                bcount = 0;
                //if got a GPS receiver
```

```
            if (getpixel(n[k].p.x,n[k].p.y) = = 6)
            {
                    n[k].fl = 1;
                    n[k].o = 0;
            }
            //if node does not have a GPS receiver
            else if (getpixel (n[k].p.x,n[k].p.y) = = 7)
            {
                    int ax = 0,ay = 0;
                    for (int i1 = n[k].p.x-n[k].range;i1<= n[k].p.x+n[k].range;i1 + +)
                    {
                            if (i1 < 0||i1 > 640)
                            continue;
                    for (int j1=n[k].p.y-n[k].range;j1<=n[k].p.y+n[k].range;j1++)
                            {
                                    if (j1 < 0||j1 > 480)
                                    continue;
                                    if (getpixel(i1,j1) = = 6)
                                    {
                                            n[k].b[ax].x = (float)i1;
                                            ax + +;
                                            n[k].b[ay].y = (float)j1;
                                            ay + +;
                                            bcount + +;
                                    }
                            }
                    }
                    if (bcount >= 3)
                    {
                            n[k].bc = bcount;
                            n[k].fl = 0;
                            n[k].o = 0;
                    }
                    else
                    {
                            n[k].fl = 0;
                            n[k].o = 1;
                    }
             }
        }
}

/* This is the regular trilateration performed on the set of random nodes
  deployed*/
```

```
point trilaterate (point p1,point p2,point p3,float d1,float d2,float d3)
{
        float i1 = p1.x,i2 = p2.x,i3 = p3.x;
        float j1 = p1.y,j2 = p2.y,j3 = p3.y;
        float x,y,w1,w2,w3;
        if (d1!= 0 && d2 != 0 && d3 != 0)
        {
                w1 = (1/(d1*d1))/((1/(d1*d1))+(1/(d2*d2))+(1/(d3*d3)));
                w2 = (1/(d2*d2))/((1/(d1*d1))+(1/(d2*d2))+(1/(d3*d3)));
                w3 = (1/(d3*d3))/((1/(d1*d1))+(1/(d2*d2))+(1/(d3*d3)));
                x = (w1*i1)+(w2*i2)+(w3*i3);
                /*if(d1!= d2 && d2 != d3)
                {
                        x   =   fabs((((2*j3-2*j2)*((d1*d1-d2*d2)+(i2*i2-i1*i1)+(j2*j2-j1*j1))-
                        (2*j2-2*j1)*((d2*d2-d3*d3)+(i3*i3-i2*i2)+(j3*j3-j2*j2)))/
                        ((2*i2-2*i3)*(2*j2-2*j1)-(2*i1-2*i2)*(2*j3-2*j2))));*/
                        ep.x = x;
                        y = (w1*j1)+(w2*j2)+(w3*j3);
                        //y     =     fabs(((d1*d1-d2*d2)+(i2*i2-i1*i1)+(j2*j2-j1*j1)+x*(2*i1-
                        2*i2))/(2*j2-2*j1));
                        ep.y = y;
                }
        return ep;
}

/* This function is used to perform trilateration on square points deployed
   assuming that there is a node at those points*/

void    max_trilaterate    (float    mxb[30][130],float    myb[30][130],float    mxe[130],float
mye[130],float d1,float d2,float d3,int mp)
{
        float i1 = mxb[0][mp],i2=mxb[1][mp],i3=mxb[2][mp];
        float j1 = myb[0][mp],j2=myb[1][mp],j3=myb[2][mp];
        float x,y,w1,w2,w3;
        if(d1 != 0 && d2 != 0 && d3 != 0)
        {
                w1 = (1/(d1*d1))/((1/(d1*d1))+(1/(d2*d2))+(1/(d3*d3)));
                w2 = (1/(d2*d2))/((1/(d1*d1))+(1/(d2*d2))+(1/(d3*d3)));
                w3 = (1/(d3*d3))/((1/(d1*d1))+(1/(d2*d2))+(1/(d3*d3)));
                x = (w1*i1)+(w2*i2)+(w3*i3);
                mxe[mp] = x;
                y = (w1*j1)+(w2*j2)+(w3*j3);
                mye[mp] = y;
        }
}
```

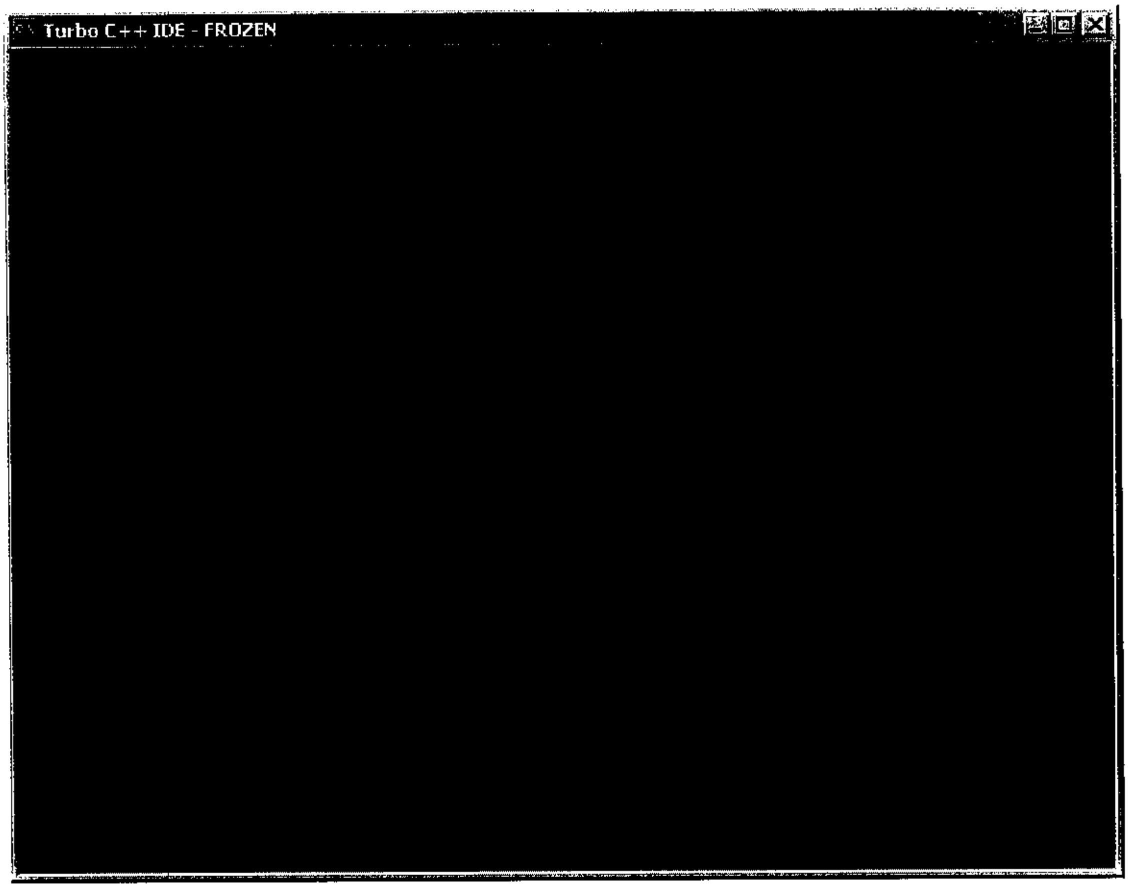
```
/* called by the graphics kernel to allocate memory */

void far * far _graphgetmem (unsigned size)
{
        // printf ("_graphgetmem called to allocate %d bytes.\n", size);
        // printf ("press any key:");
        //getch ();
        //printf ("\n");

        /* allocate memory from far heap */
        return farmalloc (size);
}

/* called by the graphics kernel to free memory */

void far _graphfreemem (void far *ptr, unsigned size)
{
        //printf ("_graphfreemem called to free %d bytes.\n", size);
        //printf ("press any key:");
        //getch ();
        //printf ("\n");

        /* free ptr from far heap */
        farfree (ptr);
}
```

SAMPLE OUTPUT

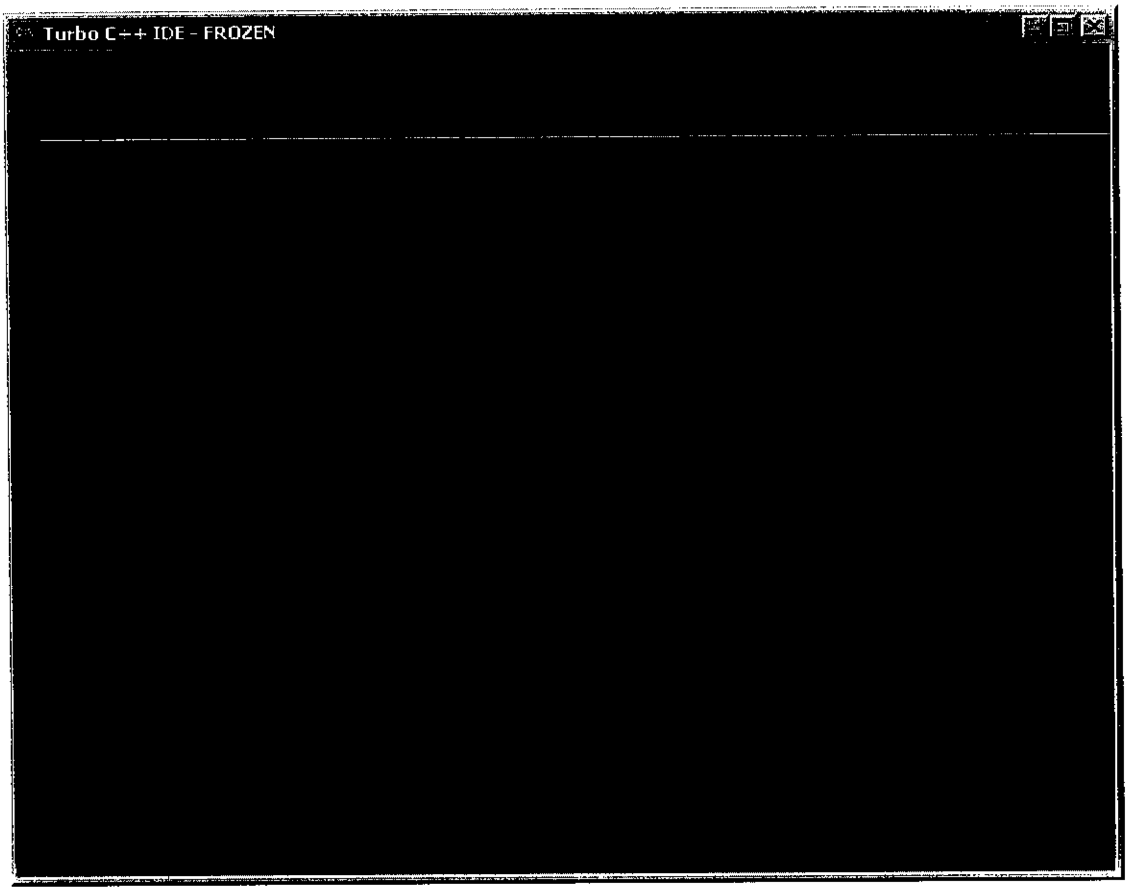
B. Sample Output

ACCEPTING INPUT FROM THE USER

DEPLOYING NODES AND PERFORMING TRILATERATION



RESULT DISPLAY



THE COPY OF PAPER PRESENTED IN WOCN 2006 CONFERNCE

April 11, 12 and 13, 2006
Bangalore, India

Conference location at Le Meridien Hotel, Bangalore, INDIA

The third IEEE and IFIP International Conference on wireless and Optical Communications Networks (WOCN 2006) invites high-quality recent research results in the areas of Mobile and Wireless Communications, optical communications and networking, architectures, protocols, planning and design, management and operation. The main goal of the conference is to bring together scientists and engineers who work on wireless and optical communications networks. WOCN2006 is co-organized by IEEE Bangalore Section, Universities in India and sponsored by IFIPTC6 WG6.8 (Mobile and Wireless Communications), WG6.6 (Management of Networks and Distributed Systems), WG6.10 (Photonic Networking), IEEE ComSoc Technical Committee on Information Infrastructure (TCII), IEEE, IEEE Communications Society and Technical Committee on Personal Communications (TCPC)

IEEE

IEEE
COMMUNICATIONS
SOCIETY

WOCN 2006 Executive Committee

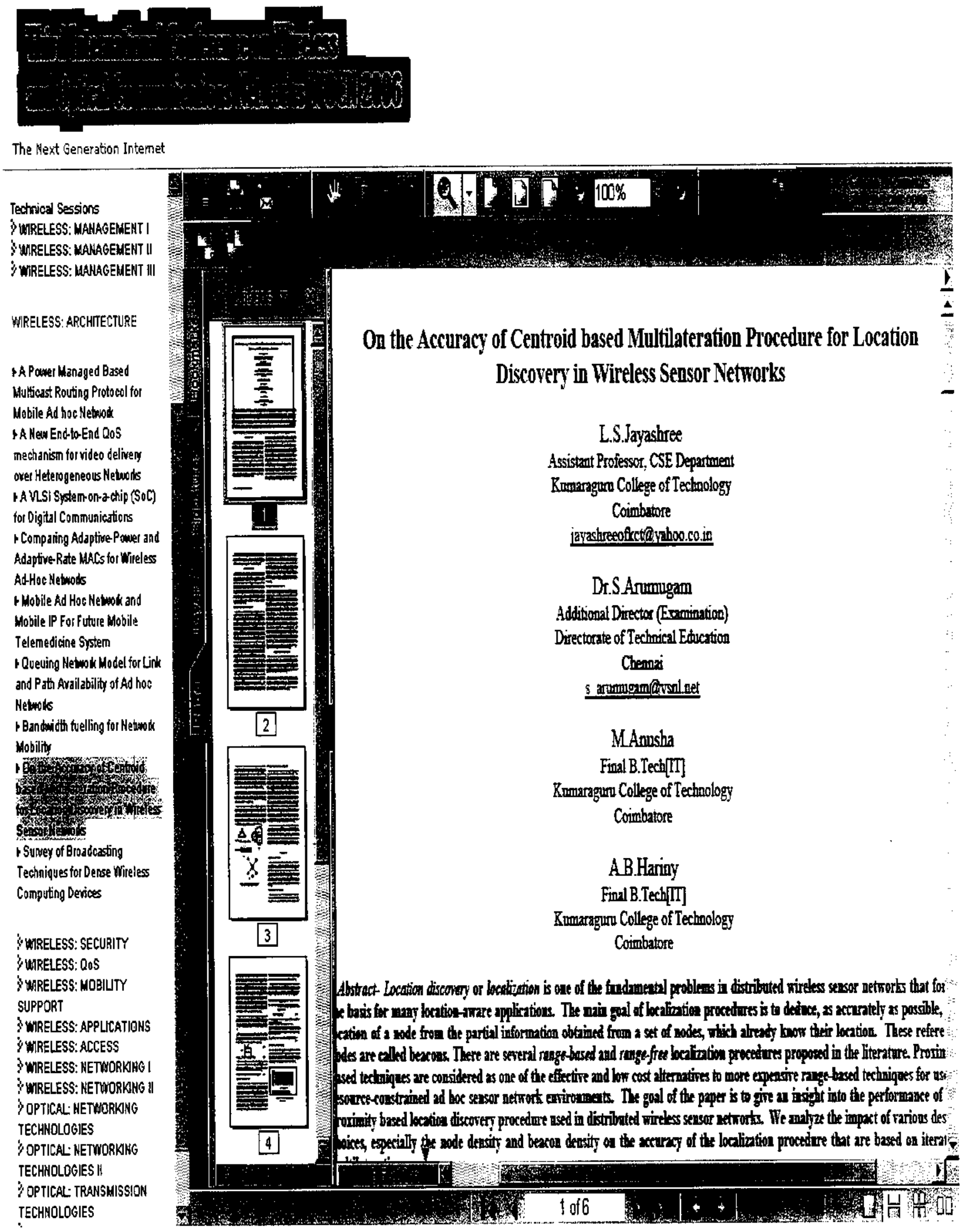
rganizing Committee

onference Chairs:
eneral Chair: Professor Guy Omidyar, Omidyar-Institute, USA
eneral co-chair: Dr. Surendra Pal, ISRO Satellite Centre, India
eneral Vice co- chairs: Professor K. Thyagarajan, IIT Delhi, India,
r. Canchi Radhakrishna, KYOCERA, USA
echnical Program co-Chairs:
rofessor Pedro Cuenca, Universidad de Castilla-La Mancha, Spain
rofessor Muhammad Jaseemuddin, Ryerson University,
rofessor Alok Kumar Das, Jadavpur University,
utorials Chairs:
r. Guy Omidyar
r. Naser-Nick Manochehri, SQU, Oman
r. Nazar Elfadil, SQU, Oman

ocal Organising Committee
hair Local Arrangement: Mr. Hitesh Mehta IEEE Bangalore Section and Eagle Photonics
ousalya, Sri Krishna College of E&T, Coimbatore , India Gopi krishna Garge, Exocore, India

dvisory and Steering Committee
anchi Radhakrishna, KYOCERA, USA
urendra Pal, IEEE Bangalore section, India
V Raghavan, IIT Madras, Chennai, India
ugusto Casaca, IST/INESC, Portugal
oon Sain Yeo, Singapore
esa M. Bastaki, UAE
amid Aghvami, UK
uy Omidyar, IFIPTC6 WG6.8 chair, USA

Technical Program Committee



The Next Generation Internet

100%

On the Accuracy of Centroid based Multilateration Procedure for Location Discovery in Wireless Sensor Networks

L.S.Jayashree
Assistant Professor, CSE Department
Kumaraguru College of Technology
Coimbatore
jayashreeofkct@yahoo.co.in

Dr.S.Arumugam
Additional Director (Examination)
Directorate of Technical Education
Chennai
s_arumugam@vsnl.net

M.Anusha
Final B.Tech[IT]
Kumaraguru College of Technology
Coimbatore

A.B.Hariny
Final B.Tech[IT]
Kumaraguru College of Technology
Coimbatore

Abstract- Location discovery or *localization* is one of the fundamental problems in distributed wireless sensor networks that for
e basis for many location-aware applications. The main goal of localization procedures is to deduce, as accurately as possible,
cation of a node from the partial information obtained from a set of nodes, which already know their location. These refere
odes are called beacons. There are several *range-based* and *range-free* localization procedures proposed in the literature. Proxim
sed techniques are considered as one of the effective and low cost alternatives to more expensive range-based techniques for us
source-constrained ad hoc sensor network environments. The goal of the paper is to give an insight into the performance of
roximity based location discovery procedure used in distributed wireless sensor networks. We analyze the impact of various des
hoices, especially the node density and beacon density on the accuracy of the localization procedure that are based on itera

1 of 6

On the Accuracy of Centroid based Multilateration Procedure for Location Discovery in Wireless Sensor Networks

L.S.Jayashree
Assistant Professor, CSE Department
Kumaraguru College of Technology
Coimbatore
jayashreeofkct@yahoo.co.in

Dr.S.Arumugam
Additional Director (Examination)
Directorate of Technical Education
Chennai
s_arumugam@vsnl.net

M.Anusha
Final B.Tech[IT]
Kumaraguru College of Technology
Coimbatore

A.B.Hariny
Final B.Tech[IT]
Kumaraguru College of Technology
Coimbatore

Abstract- *Location discovery* **or** ***localization*** **is one of the fundamental problems in distributed wireless sensor networks that forms the basis r many location-aware applications. The main goal of localization procedures is to deduce, as accurately as possible, the location of a node om the partial information obtained from a set of nodes, which already know their location. These reference nodes are called beacons. ere are several *range-based* and *range-free* localization procedures proposed in the literature. Proximity based techniques are considered as e of the effective and low cost alternatives to more expensive range-based techniques for use in resource-constrained ad hoc sensor twork environments. The goal of the paper is to give an insight into the performance of the proximity based location discovery procedure ed in distributed wireless sensor networks. We analyze the impact of various design choices, especially the node density and beacon nsity on the accuracy of the localization procedure that are based on iterative multilateration.**

Keywords- **Ad hoc sensor networks, Localization, Proximity based localization, Beacons, Localization accuracy, Resource efficiency**

I.INTRODUCTION

Wireless Ad Hoc Sensor Networks(WASNs) are networks tiny, battery powered sensor nodes with limited on-board ocessing, storage and radio capabilities [1]. Nodes sense d send their reports toward a processing center, which is lled a base station or a "sink." Designing protocols and plications for such networks has to be energy aware in der to prolong the lifetime of the network. Sensor tworks, once deployed, are left unattended and expected to ork for extended periods of time. This is true under many al world application settings, rendering battery replacement t of question - the life of the battery decides the life of the twork. Owing to the importance of the problem, there is a gnificant body of research addressing different aspects of wer control problem. [1] gives a detailed survey on sensor tworks and the open research problems.

In sensor networks, nodes are deployed into an unplanned infrastructure where there is no *a priori* knowledge of location. The problem of estimating spatial-coordinates of the node is referred to as localization. An immediate solution that comes to mind is GPS or the Global Positioning System. However, there are some strong factors against the usage of GPS. For one, GPS can work only outdoors. Secondly, GPS receivers are expensive and not suitable in the construction of small cheap sensor nodes. A third factor is that it cannot work in the presence of any obstruction like dense foliage etc. Thus, sensor nodes would need to have other means of establishing their positions and organizing themselves into a coordinate system without relying on an existing infrastructure. The main practical objective is to locate each node as accurately as possible with a given information with a certain amount of error about the distances between a subset of nodes.[2]. [13] presents a survey on the various location discovery techniques in wireless ad hoc sensor networks.

he remainder of the paper is organized as follows: section
iscusses the related work. Section III presents a discussion
the three most-widely used localization techniques.
ction IV gives details of implementation and discussed the
ults obtained. Finally, section V concludes the paper

II. RELATED WORK:

Ve present a brief summary of the current location
covery techniques and systems. We restrict our attention
ly to the techniques used in WASNs. With regard to the
chanisms used for estimating location, we divide these
alization protocols into two categories: *range-based* and
ge-free. [range-free loc.sys]. The former is defined by
otocols that use absolute point-to-point distance estimates
nge) or angle estimates for calculating location. The latter
kes no assumption about the availability or validity of
ch information. Because of the hardware limitations of
ASN devices, solutions in range-free localization are being
rsued as a cost-effective alternative to more expensive
ge-based approaches.

Range-Based Localization Schemes

Time of Arrival (TOA) technology is commonly used as a
eans of obtaining range information via signal propagation
ne. The most basic localization system to use TOA
hniques is GPS [3]. GPS systems require expensive and
ergy-consuming electronics to precisely synchronize with a
ellite's clock. With hardware limitations and the inherent
ergy constraints of sensor network devices, GPS and other
)A technology present a costly solution for localization in
reless sensor networks. Time *difference of arrival (TDoA)*
hnique is similar to the TOA method. The only difference
in this approach the system assumes the presence of
nultaneously emitted signals from two beacons and finds
time difference between the two signals. While many
frastructure-based systems have been proposed that use
)OA [5][6], additional work such as AHLos [7][8] has
ployed such technology in infrastructure-free sensor
tworks. Like TOA technology, TDoA also relies on
tensive hardware that is expensive and energy consuming,
aking it less suitable for low-power sensor network devices.
gle of arrival (AoA) techniques measure the angle between
number of beacons (more than three) and an object to
termine the position of the object. The precision of the
A techniques diminishes with increasing the distances
tween the unknown object and the beacons [9]. *Received
nal strength indicator (RSSI)* techniques works by
serving the power of the received signal. Assuming that
original power of the received signal at a transmitter is
own, the propagation loss can be used to estimate the
stance between the transmitter and the receiver. The
stem should have a map between the loss and the distances,
ich is often either an empirical table or an equation-based
odel. The errors in this type of distance measurement can be quiet high, due to the obstacles and the multipath effects of the environment on the signal. This technique is mostly implemented with the radio frequency (RF) transmitter and receivers, and has been used with the GSM technology. [4] and [11] make use of such techniques. While solutions based on RSSI have demonstrated their efficacy in simulation and in a controlled laboratory environment, the premise that distance can be determined based on signal strength, propagation patterns, and fading models remains questionable, creating a demand for alternate localization solutions that work in resource constrained environments are warranted.

B. Range-Free Localization Schemes

In sensor networks and other distributed systems, errors can often be masked through fault tolerance, redundancy, aggregation, or by other means. Depending on the behavior and requirements of protocols using location information, varying granularities of error may be appropriate from system to system. Acknowledging that the cost of hardware required by range-based solutions may be inappropriate in relation to the required location precision, researchers have sought alternate range-free solutions to the localization problem in sensor networks. In [12], a heterogeneous network containing powerful nodes with established location information is considered. In this work, anchors beacon their position to neighbors that keep an account of all received beacons. Using this proximity information, a simple centroid model is applied to estimate the listening nodes' location. We refer to this protocol as the *Centroid algorithm.* An alternate solution, *DV-HOP* [13] assumes a heterogeneous network consisting of sensing nodes and anchors. Instead of single hop broadcasts, anchors flood their location throughout the network maintaining a running hop-count at each node along the way. Nodes calculate their position based on the received anchor locations, the hop-count from the corresponding anchor, and the average-distance per hop, a value obtained through anchor communication.

III. LOCALIZATION TECHNIQUES

We now describe in detail the three major classes of localization algorithms. These algorithms operate on an ad-hoc network of sensor nodes where a small percentage of the nodes are aware of their positions either through manual configuration or using GPS. Though these techniques are mainly associated to range-based techniques, they also find their use with range-free techniques too.

Once we measure the distances between an object and a number of beacons, we also need a way to combine the measurements to find the actual position. The most common methods to combine the distance measurements from three or more beacons are *triangulation, simple trilateration* (multilateration), and *atomic multilateration.*

Triangulation is a method for finding the position of a node, ıen the angles are measured by AoA technique. An ample of such a procedure is shown in Figure 2.a. The ject *X* measures its angles with respect to the beacons *A*1, ! and *A*3. The measured angles form three straight lines ɔng the directions *XA*1, *XA*2 and *XA*3. The intersection tween the three lines defines the location of the node *X*. ıe accuracy of this technique is heavily dependent upon the curacy of the employed angle measurement technique.

Simple trilateration is used when we have an accurate timate of distances between a node and at least three acon nodes. This simple method finds the intersection of ree circles centered at beacons as the position of the node. ıe scenario is shown in Figure 2.b.

Atomic multilateration is accepted as the most appropriate ay to determine the location of a sensor node based on cations of beacons. An example is shown in Figure 2.c, here the nodes *A*1, *A*2, *A*3, and *A*4 are beacons, with ıown estimates of their locations, while the node *X* timates its location using a multilateration procedure. The ocedure attempts to estimate the position of a node by inimizing the error and discrepancies between the measured ılues.

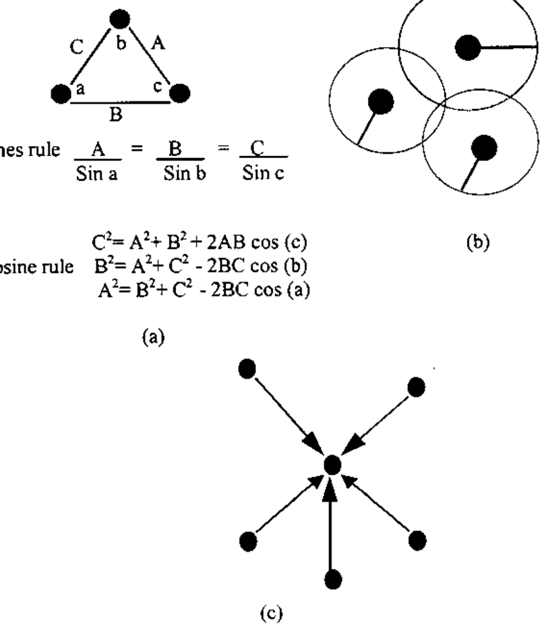

C
b
A
a
c
B
ıes rule
$\frac{A}{\text{Sin a}} = \frac{B}{\text{Sin b}} = \frac{C}{\text{Sin c}}$
$C^2 = A^2 + B^2 + 2AB\cos(c)$
ɔsine rule $B^2 = A^2 + C^2 - 2BC\cos(b)$
$A^2 = B^2 + C^2 - 2BC\cos(a)$
(a)
(b)
(c)

Figure 2: Localization Basics a) Triangulation b) Trilateration c) Atomic Multilateration

Proximity-based LocalizationTtechnique

Most of the proposed localization techniques today, depend on recursive trilateration/multilateration techniques. One way of considering sensor networks is taking the network to be organized as a hierarchy with the nodes in the upper level being more complex and already knowing their location through some technique (say, through GPS). These nodes then act as beacons by transmitting their position periodically. The nodes that have not yet inferred their position listen to broadcasts from these beacons and use the information from beacons with low message loss to calculate its own position. A simple technique would be to calculate its position as the centroid of all the locations it has obtained. This is called as *proximity based localization.* It is quite possible that all nodes do not have access to the beacons. In this case, the nodes, which have obtained their position through proximity, based localization themselves act as beacons to the other nodes. This process is called *iterative multilateration.* As can be guessed, iterative multilateration leads to accumulation of localization error since the position estimated for the unknown node depends upon the previous estimations which themselves are not accurate values.

Thus the focus of this paper is to analyze the impact of various design choices, especially the node density, beacon density and the range of each beacon on the accuracy and the effectiveness of the localization procedure. We describe accuracy in terms of the average localization error and effectiveness in terms of the number of the nodes that could resolve their position when the procedure convergences. (i.e., the point after which no further improvement is evidenced).

IV. ANALYSIS

A.Simulation Environmet:

We assume that simulations are performed in an *75* X *50 m* rectangular area. Nodes are randomly placed over this region. The real locations of sensor nodes *Ai*, i=0,..,n are represented as points *Ai*(*xi; yi*). Coordinates *xi* and *yi* are generated from two uniform distributions, one on the interval [0,*Xmax*] and one on the interval [0,*Ymax*]. For each simulation, a subset of nodes that have initial estimates of their locations is randomly or according to user specified criteria selected. The transmission range of each beacon is set to 20m.

B.The Localization Procedure

We refer to the nodes with known positions as beacon nodes and those with unknown positions as unknown nodes.

r goal is to estimate the positions of as many unknown des as possible in a fully distributed fashion. The location covery algorithm used for this analysis is a typical ample for the class of localization technique that follow rative multilateration procedure. Centroid algorithm that ls in this category, though simple, has been proven to be rly effective for use with resourcec-constrained wireless nsor networks. Once the sensor network is deployed, the acon nodes broadcast their locations to their neighbors. ighboring unknown nodes measure their separation from eir neighbors and use the broadcasted beacon positions to imate their own positions. Once an unknown node imates its position, it becomes a beacon and broadcasts its imated position to other nearby unknown nodes, enabling em to estimate their positions. This process repeats until all e unknown nodes that satisfy the requirements for ultilateration obtain an estimate of their position.

Beacons situated at known positions, (*Xi* , *Yi*), transmit riodically with a time period *T*. Clients listen for a period evaluate connectivity. If the percentage of messages ceived from a beacon *Bi* with range *ri* in a time interval *t* ceeds a threshold, that beacon is considered connected at *ri*.

When the beacon placement is uniform, the centroid of the sitions of all connected beacons is a feasible solution in the gion of connectivity overlap. A client estimates its position *Kest, Yest*) to be the centroid of the positions of all nnected beacon using the following method from (bulusu):

$$Wi = \frac{\left(1/r_i^{\,2} \right)}{\left(\Sigma_{i=1}^{k} 1/r_i^{\,2} \right)} \qquad (1)$$

$$X_{est}, Y_{est)} = (\Sigma_{i=1}^{k} W_i X_i, \Sigma_{i=1}^{k} W_i X_i) \qquad (2)$$

Given the actual position of the client (*Xa, Ya*), we can mpute the accuracy of the localization estimate or the *calization error LEB*(*Xa, Ya*) , which is the distance etween the client's estimated and actual positions.

EB(*Xa, Ya*) = [(*Xest* − *Xa*)2 + (*Yest* − *Ya*)2]1/2 (3)

. Results and Discussions

The goal of the study is to give an insight into the erformance of the multilateration based localization rocedure under different design choices. At the time of eployment of a sensor network to accomplish a given iission, there are a number of design parameters that are to e properly decided upon so as to get an optimum level of erformance. There are many tunable parameters left to the hoice of the network designer, like the node density, beacon ensity, beacon range etc (only w.r.t. to localization echniques, for brevity). These heuristics are chosen either ased upon the application's optimality requirements or decided empirically by the network designer. In this study, we try to estimate the impact of the various design choices on the accuracy and efficacy of the localization procedure.

We generate 50 different topology and the results shown in graphs are the averages of the values measured in each simulation run. We start by considering the impact of beacon density on the accuracy of the localization procedure. We use the following measures for the evaluation of the algorithm:

Beacon density(ρ)- denotes the number of beacons deployed per unit area

BPRN(Beacons Per Radio Neighbourhood)- denotes the number of beacons available per radio range of a node

$$BPRN = \rho.\Pi.Range^{2} \qquad (4)$$

For instance, Fig. 3 shows the values of BPRN calculated for varying beacon densities.

ARLE(Average Relative Localization Error)

This is the ratio of the average localization error of all nodes measured with n beacons to that with 3 beacons (a minimum number needed for trilateration). ARLE is calculated using the following equation:

$$[\Sigma_{i=1}^{N} \sqrt{(Y_{i0}^{n}-Y_{i0})^2+(X_{i0}^{n}-X_{i0})^2/(Y_{i0}^{3}-Y_{i0})^2+(X_{i0}^{3}-X_{i0}})^2]/N \qquad (5)$$

Beacon density VsEfficiency

We first analyze the impact of beacon density on the usefulness of the procedure. The graphs in Fig 3-5 illustrate how the beacon density influences the number of nodes resolved and the speed of convergence of the procedure. For this study, the node intensity is varied from 50 to 125.

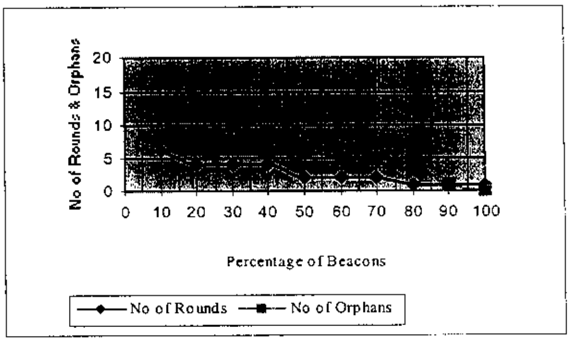


Fig 3 Number of nodes resolved for node intensity=50

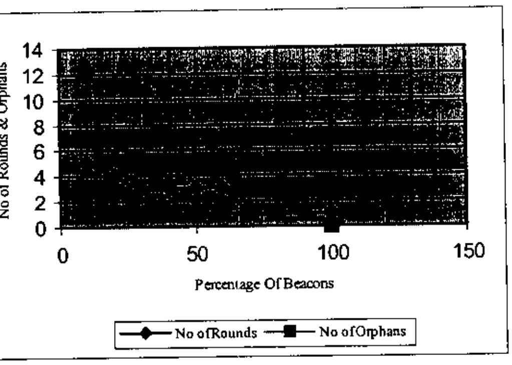


Fig 4 Number of nodes resolved for node intensity=100

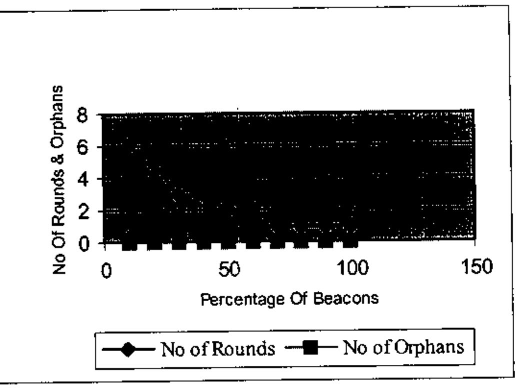


Fig 5 Number of nodes resolved for node intensity=125

s evident from these graphs, as the beacon density reases, the speed of the procedure increases and the mber of orphans(nodes remaining unresolved of eir location) decreases. Another more important servation is that, as the node intensity increases, the *ocedure converges very quickly* because of the reased connectivity among the nodes

acon density Vs. Accuracy

e now proceed to analyze the impact of beacon nsity on the accuracy of the localization procedure, easured in terms of ARLE.

50 nodes

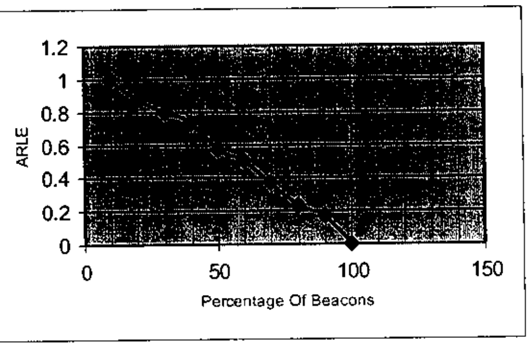


Fig 6 Localization Accuracy

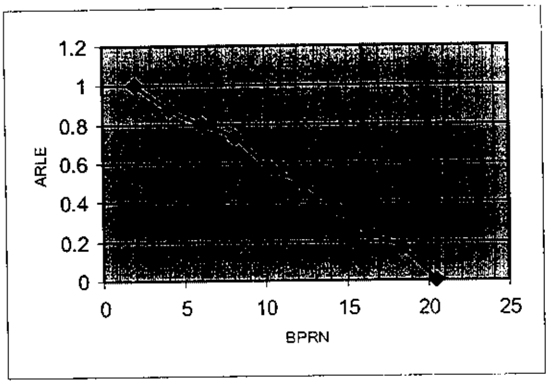


Fig. 7 Localization Accuracy w.r.t. BPRN

100 nodes

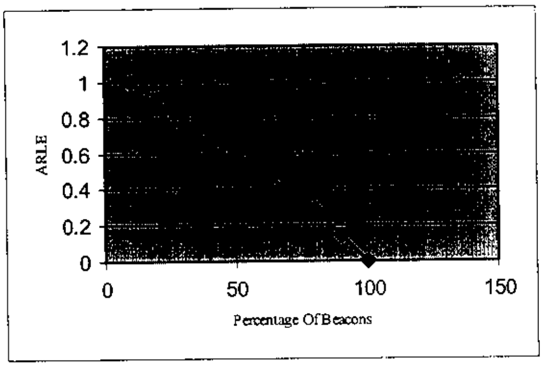


Fig.8. Localization Accuracy

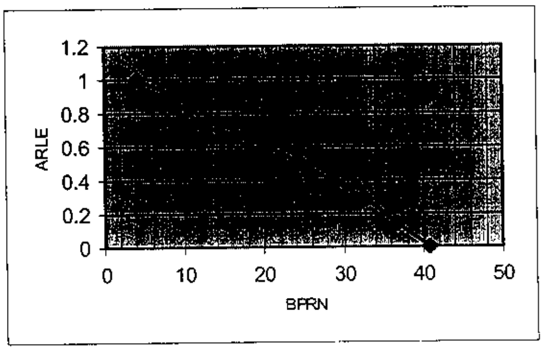


Fig.9. Localization Accuracy w.r.t. BPRN

125 nodes

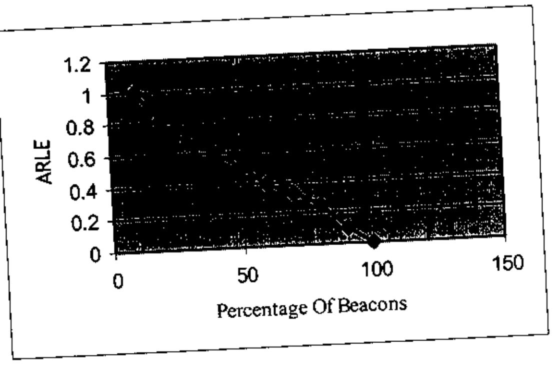


Fig 10 Localization Accuracy

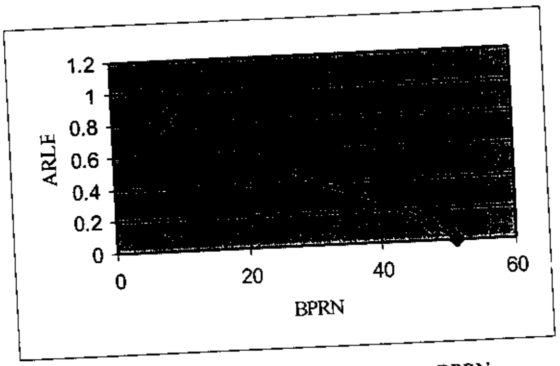


Fig 11 Localization Accuracy w.r.t. BPRN

.6-11 show how the localization error decreases as and more number of beacons are deployed. But cally, the trade off between the cost of the network and curacy of the locations discovered should be considered :iding the optimum beacon density.

V. Conclusion

rative multilateration procedures play an important role cation discovery problem in ad hoc wire less sensor orks. Centroid or proximity based technique is one such iple: given the inherent resource constraints of WASNs the desired localization accuracy of the intended cation, it is regarded as a cost effective and sufficient iative for more expensive range based techniques.

The distinguished advantage of this Centroid localization scheme is its simplicity and ease of implementation. From our extensive study, we analyzed the impact of one of the most important architectural parameter, viz. the beacon density on the accuracy and effectiveness of the localization procedure. From this study, we conclude that, though this technique is coarse-grained leading to significant localization errors, may prove useful for certain non-critical applications.

References

[1] Ian F.Akylidiz, Shankarasubramaniam,"A Survey on Sensor Networks," *IEEE Communications Magazine*, August 2002.

[2] N. Bulusu, J. Heidemann and D. Estrin., " Self-Configuring Localization Systems: Design and Experimental Evaluation", ACM Transactions on Embedded Computing Systems, Vol. 3, No. 1, February 2004, Pages 24–60.

[3] B. H. Wellenhoff, H. Lichtenegger and J. Collins, *Global Positions System: Theory and Practice*, Fourth Edition. Springer Verlag, 1997.

[4] P. Bahl and V. N. Padmanabhan, "RADAR: An In-Building RF-Based User Location and Tracking System", In *Proceedings of the IEEE INFOCOM '00*, March 2000.]

[5] A. Harter, A. Hopper and P. Steggles, A. Ward and P. Webster, "The anatomy of a context-aware application", In *Proceedings of MOBICOM '99*, Seattle, Washington, 1999

[6] N. B. Priyantha, A. Chakraborty and H. Balakrishnan, "The Cricket Location-Support System", In *Proceedings of MOBICOM '00*, New York, August 2000.

[7] A. Savvides, C. C. Han and M. B. Srivastava, "Dynamic Fine- Grained Localization in Ad-Hoc Networks of Sensors", In *Proceedings of MOBICOM '01*, 2001, Rome, Italy, July 2001.]

[8] A. Savvides, H. Park and M. Srivastava, "The Bits and Flops of the N-Hop Multilateration Primitive for Node Localization Problems", In *First ACM International Workshop on Wireless Sensor Networks and Application*, Atlanta, GA, September 2002.

[9] D. Niculescu and B. Nath,"Ad Hoc Positioning System (APS) using AoA", *INFOCOM' 03*, San Francisco, CA,2003

[10] J.Hightower, G. Boriello and R. Want, SpotON: An indoor 3D Location Sensing Technology Based on RF Signal Strength, *University of Washington CSE Report #2000-02-02*, February 2000

[11] N. Bulusu, J. Heidemann and D. Estrin, "GPS-less Low Cost Outdoor Localization for Very Small Devices", *IEEE Personal Communications Magazine*, 7(5):28-34, October 2000.

[12] D. Niculescu and B. Nath, "DV Based Positioning in Ad hoc Networks", In *Journal of Telecommunication* Systems, 2003.

[13] F.Koushanfar et al. "Location Discovery in Ad-hoc Wireless Sensor Networks", *Ad Hoc Wireless Networking*, Kluwer Academic Publishers, 2003.

[14] Tian He, Chengdu Huang, Brian M. Blum, John A. Stankovic, Tarek Abdelzaher." Range-Free Localization Schemes for Large Scale Sensor Networks, *MobiCom '03*, September, 2003

REFERENCES

0. REFERENCES

1. N. Bulusu, J. Heidemann and D. Estrin., "Adaptive Beacon placement", in the Proceedings of the 21st International Conference on Distributed Computing Systems, Phoenix, Arizona, April 2001.
2. Seapahn Meguerdichian, Sasa Slijepcevic, Vahag Karayan, Miodrag Potkonjak , "Localized algorithms in wireless ad-hoc networks", in proceedings of ACM, May 2001.
3. Savvides, C. C Han and M. B. Srivastava, "Dynamic Fine-Grained Localization in Ad-Hoc Networks of Sensors", In Proceedings of MOBICOM '01, 2001, Rome, Italy, July 2001.
4. N. Bulusu, J. Heidemann and D. Estrin., "Self configuring Localization systems: Design and Experimental Evaluation", ACM Transactions on Embedded Computing Systems, Vol. 3, No. 1, February 2004.
5. L.S.Jayashree, Dr.S.Arumugam, M.Anusha, A.B.Hariny "On the Accuracy of Centroid based Multilateration Procedure for Location Discovery in Wireless Sensor Networks", proceedings of the Third IEEE and IFIP Conference on Wireless and Optical Networks (WOCN 2006), April 2006
6. The C++ Programming Language, Bjarne Stroustrup. Addison-Wesley, 2nd edition, 1991.
7. Object Oriented Programming with C++, Robert Lafore, Galgotia Publicatons Private Ltd, third edition,1999.
8. Programming with C++, K.R.Venugopal, Sudeep.R.Prasad, TMH outline Series,2003.