

GDL: A Geographic Distributed Localization Algorithm for Wireless Sensor Networks

Yurong Xu

James Ford

Fillia S. Makedon

Dept. of Computer Science and Dartmouth Experimental Visualization Laboratory (DEVLAB)

Dartmouth College,

Hanover NH 03755, USA

{yurong, jford, makedon}@cs.dartmouth.edu

Abstract— Localization is a fundamental problem in sensor networks. This paper proposes a new distributed localization algorithm called GDL (“Geographic Distributed Localization”) that is based on hop-counting. Although the idea of predicting node position from connectivity information in WSNs is not new, our algorithm improves on other localization algorithms of this type in several key ways. Results from extensive simulation in NS-2 show a significant improvement in the accuracy with which locations can be determined in WSNs with varying density and different number of nodes on irregular placements. In particular, when compared with the MDS-MAP series of algorithms, our algorithm achieves about a 30–40% improvement in accuracy. It also has a lower communication cost $O(n)$ versus $O(n \log n)$ than MDS-MAP(P), and has a low constant memory cost per node.

Keywords-component; Position estimation, localization, sensor networks, hop counting, multidimensional scaling.

I. INTRODUCTION

Wireless Sensor Networks (WSNs) [1,2], an emerging new type of ad-hoc network, integrate sensing, processing and wireless communication in a distributed system. WSNs have numerous applications, such as surveillance, healthcare, industry automation, and military uses.

Localization is a fundamental problem in a sensor network. There are many location-aware applications of WSNs in which it is necessary to know the location of nodes. In addition, geographic routing, which is important in many WSNs [3], requires that each node be aware of its location.

As described in more detail in Section 2, there are many physical features in WSNs that have been utilized by different localization algorithms to generate the location of nodes. Based on the type of features used in localization, localization algorithms are divided into two categories [12]: range-based and range-free. While producing fine-grained locations, range-based algorithms remain cost-ineffective due to the cost of hardware or the strict requirements on time synchronization.

The category of range-free approaches relies mainly on hop-counting and connectivity. Coarse-grained range-free protocols are relatively easy to develop because the utilized features are inherent in any WSN without any additional hardware. There are a number of algorithms that use hop or connectivity information for localization. Some of them are based on the idea of letting nodes derive their position in terms of connectivity to some anchor nodes (for which there is pre-calculated physical geographic information) [4, 5]. However, an obvious weakness in relying on this approach is that nodes will be completely uncertain about their real coordinates in the absence of anchor nodes.

Algorithms without anchor nodes have been proposed to address this point, but extensive simulation experiments on different localization algorithms [4, 7, 15, 16] based only on hop-counting or connectivity have shown that there is usually a sizeable error in the location generated by these algorithms in comparison with actual locations, even after some refinement processes such as the ones in [15] and [16].

In this paper, we propose a new algorithm—GDL, or “Geographic Distributed Localization”—that generates a more accurate location for each node based only on hop-counting and without anchor nodes. We show how a distributed application of this algorithm can be made with constant costs in terms of the amount of memory, computation and communication overhead required per node and $O(n)$ for an overall n -node network. Based on extensive simulations in NS-2, our localization algorithm shows accuracy improvements of about 45% and 36%, respectively, in effective range, compared with the localization algorithms MDS-MAP [15] and MDS-MAP(P) [16], which outperform the other existing localization algorithms of this type under these conditions [15,16].

The paper is organized as follows: Section 2 summarizes related work, Section 3 introduces our localization algorithm, Section 4 describes the simulations used to test our method, and a final section gives our conclusions.

II. RELATED WORK

Node localization in WSNs has been an active research field for some time and there are many algorithms that have been proposed to set real coordinates in WSNs. As mentioned

above, we will roughly separate methods into two categories: range-based and range-free.

A. Range-based Methods

Range-based protocols use absolute point-to-point distance or angle information to calculate the location between neighboring sensors. One common technique for distance/angle estimation is to use angle of arrival (AOA) in addition to hop information [17]; similar methods use time of arrival (TOA) [19], time difference of arrival (TDOA) [11], or Received Signal Strength Indicator (RSSI) [8] in addition to hop information. As noted above, AOA, TOA, and TDOA all have additional hardware requirements beyond what is needed for basic WSN functions. AOA requires additional devices such as ultrasound transceivers [8] or directional antennas [9], and TOA and TDOA requires nodes to have two different communication devices with unequal propagation speeds, such as ultrasound transceivers and RF transceivers [18], to measure time differences. The utilization of RSSI depends on a direct relationship between the distance and the signal strength, but this relationship is affected unpredictably by the antenna, battery, electric components inside of the node, and the outside environment [8]. In summary, range-based protocols are frequently cost-ineffective as alternatives for producing fine-grained locations due to the requirement of additional hardware, the strict requirements on time synchronization this hardware entails, and the resulting increase in energy consumption.

B. Range-free Methods

Typical approaches to avoiding range hardware, such as those described in [4, 15, 16], make use of the connection information of the network—something inherent in any WSN without any additional hardware. Some approaches are based on the idea of letting nodes derive their position in terms of connectivity to special anchor nodes (which have predetermined geographic information) [4]. However, an obvious weakness in relying on this approach is that nodes will be completely uncertain about their real coordinates in the absence of anchor nodes. Our approach is one of the class designed to operate without anchor nodes.

Shang et al. describe an algorithm called MDS-MAP(P) [16] that extends MDS-MAP[15] into a distributed algorithm. MDS-MAP(P) first computes local maps for each node with MDS using local shortest paths, then merges local maps into a global map. MDS-MAP(P) outperforms most other algorithms and has a computation cost of $O(k^3n)$ (here k is the number of neighbor nodes). Even if the algorithm doesn't include its optional refining step, it still needs at least $O(n\log(n))$ communication cost based on using a binary aggregation tree to route messages in order to send back the local map from each local node. Another drawback of MDS-MAP(P) is that the memory requirement per node will be $O(n)$ in order to store the $O(n)$ global map when the local map merging process is being completed. In our algorithm, in contrast, we require only $O(C)$ memory per node, where C is a constant. Our algorithm achieves more accurate results than MDS-MAP series algorithms based on same placements in simulation. Our algorithm uses a method similar to steps 2.a and 2.b in MDS-MAP(P) for setting coordinates of nodes based on pair

distances, since the MDS algorithm is an efficient algorithm for this.

Authors in [7] proposed an optimized solution for the localization problem in one-dimensional Euclidean line WSNs. The idea of this algorithm is roughly similar to our approach when it operates in a 1-D environment; however, while the authors in [7] proposed a framework for extending to two or more dimensions, their work does not provide an actual 2-D localization algorithm.

III. GEOGRAPHIC DISTRIBUTED LOCALIZATION ALGORITHM

Before you begin to format your paper, first write and save the content as a separate text file. Keep your text and graphic files separate until after the text has been formatted and styled. Do not use hard tabs, and limit use of hard returns to only one return at the end of a paragraph. Do not add any kind of pagination anywhere in the paper. Do not number text heads—the template will do that for you.

Finally, complete content and organizational editing before formatting. Please take note of the following items when proofreading spelling and grammar:

A. Definition of Hop Coordinate

Before we present our algorithm, we will introduce the concept of a “*Hop Coordinate*” [20], which plays a key role in our algorithm. Unlike other localization algorithms, which only count the number of hops or track connectivity between nodes, our approach not only counts the number of hops from some bootstrap node but also offsets this count with local network information specific to a given node.

Definition 1: A Hop Coordinate is constructed from two parts: a *number of hops* and an *offset*. The first part is a positive integer which equals the number of hops in a minimum hop route from some bootstrap node to a given node. The second part can be seen as a decimal fraction generated from local network information that refines the hop number:

$$\text{hop}_A^i = \begin{cases} \min_{B \in N_A} (\text{hop}_B^i) + 1, & \text{when } |N_A| \neq 0; \\ \infty, & \text{when } |N_A| = 0. \end{cases}$$

$$\text{offset}_A^i = \begin{cases} \frac{\sum_{B \in N_A} (\text{hop}_B^i - (\text{hop}_A^i - 1)) + 1}{2(|N_A| + 1)}, & \text{when } |N_A| \neq 0; \\ 0, & \text{when } |N_A| = 0. \end{cases}$$

$$\text{Hop coordinate: } h_A^i = \text{hop}_A^i + \text{offset}_A^i$$

Here, A is a node, hop_A^i is the minimum number of hops to reach node A counting from some bootstrap node i , N_A is a set of nodes which can be reached by node A in one hop, and $|N_A|$ is the number of nodes in N_A .

In the remainder of the paper, we also use some alternate notations to represent hop coordinates: h_A means the hop coordinate of node A with respect to some bootstrap node, h^i means a hop coordinate with respect to bootstrap node

i , $offset_A$ means the offset of the hop coordinate of node A with respect to some bootstrap node, $offset^i$ means an offset using bootstrap node i , hop_A^i means the minimum number of hops to node A from bootstrap node i , and hop^i means the number of hops to bootstrap node i .

B. Description of the Algorithm

In order to simplify communication, we assume that WSNs are symmetric (which means that if node A can reach B , then B can reach A) and transitive. These assumptions are reasonable for our algorithm, since our algorithm is simulated above the 802.15.4 MAC layer [13] in NS-2 [10], which guarantees communication will be symmetric (in NS-2, every message transmitted in the 802.15.4 MAC layer from node A to node B will be followed up by an ACK message from node B).

Before we run our algorithm, we assume that:

- There are n nodes in the WSN, inside which there are b Bootstrap node candidates (here b is a constant; in our experiments we use $b = 8$ for less than 1000 nodes and $b = 16$ for 1000 or more nodes).
- Each node has a unique ID, which is a pre-defined number that can uniquely identify that node. Here we use digits to represent IDs. Providing a unique ID for each node is generally easy to achieve in practice, such as when nodes are pre-initialized uniquely by the manufacturer.

Our GDL algorithm can be summarized in five steps:

Step 1) : Select 2 proper bootstrap nodes from b bootstrap node candidates.

Step 1.1: Every Bootstrap node candidate sends out a message to count the hop distance from itself to every other bootstrap node candidates.

Step 1.2: After each bootstrap node candidate has a $b \times b$ pair-distance matrix for the b candidates, each candidate will compare the distance between each pair of nodes, letting the pair of nodes with the longest distance “win”. In case there are several pairs with the same longest distance, then we resolve ties by selecting nodes in order of their ID (starting with the smallest) until we have selected one linked pair as the winners. This results in the deterministic selection of two bootstrap nodes, specified as X and Y (here we can let the node with the smaller ID be X and the other one Y).

In this step we incur a computational cost $O(b)$, a memory cost $O(b^2)$, and an $O(b)$ communication cost, per node, and an $O(bn)$ constant communication cost for the whole networks (if, in the worst case, we use flooding to communicate). Because there are a fixed number b of bootstrap nodes, each has a constant cost in memory, computation, and communication.

Step 2) : Compute hop coordinates.

Step 2.1: Given two bootstrap nodes X and Y assigned in step 0, initialize with one message from each with variables $hop^X = 0$ and $hop^Y = 0$, respectively, to flood the network.

Step 2.2: Every other node records its number of hops from bootstrap nodes X and Y , and thus computes the integer part of its hop coordinate.

The detailed processes in each bootstrap node and in other nodes are as follows:

(i) In bootstrap node: A bootstrap node (X or Y) creates a message with $hop^i = 0$ ($i=X$ or Y) to flood the network. After that, the bootstrap node will drop any message that was originated by itself. (ii) In all other nodes in the WSN: The following procedure₁ runs in any node A that is not a bootstrap node to compute the hop coordinate for the node:

Procedure₁ (compute Hop Coordinate) in node_A

Input: message(hop_B^i) from node $B \in N_A$, $i=X$ or Y

Output: hop_A^i , $offset_A^i$

for each message(hop_B^i) from $B \in N_A$ and \sim TIMEOUT

if ($hop_B^i < hop_A^i$)

$hop_A^i = \min(hop_A^i, hop_B^i) + 1$;

forward (message(hop_A^i)) to MAC;

else

drop (message(hop_B^i));

endif

endfor

if ($|N_A| = 0$)

$offset_A^i = 0$;

else $offset_A^i = \frac{\sum_{B \in N_A} (hop_B^i - (hop_A^i - 1)) + 1}{2(|N_A| + 1)}$;

In this step we incur a computational cost of $O(|N_A|)$, a memory cost of $O(|N_A|)$, and a $O(1)$ communication cost, per node, with a $O(n)$ overall communication cost.

Step 3) : Generate local center nodes.

Step 3.1: Elect center node for local area.

With hop coordinates $h^X = \frac{1}{2} * i$ (i ranges from 1 to N) and $h^Y = \frac{1}{2} * j$ (j from 1 to N) as boundaries, we separate the whole WSN into many local areas. We then select a node in each local area as a center node for that area, with the idea that it will be used to take care of computation of the local map in that area. Ideally, a *center position* should be $h^X = \frac{1}{2} * i + \frac{1}{4}$ and $h^Y = \frac{1}{2} * j + \frac{1}{4}$ for a local area bounded by $\frac{1}{2} * i < h^X \leq \frac{1}{2} * i + \frac{1}{2}$ and by $\frac{1}{2} * j < h^Y \leq \frac{1}{2} * j + \frac{1}{2}$. In order to select a center node for each local area, we employ a simple voting mechanism that selects a node as near as possible to the ideal center position C (in the following text, we refer to this position as the “ideal local center”). To vote, each node A in that area delays a period of time $t = f(\sqrt{(h_A^X - h_C^X)^2 + (h_A^Y - h_C^Y)^2})$. For function $f(x)$, the simplest expression we can use is Cx , where C is a constant. In the period of time until t , the node will listen to other nodes in its area, and if no messages are received, then the node will send a message to vote itself as a center node of that local area. The whole procedure including a collision detection provision is shown in procedure₂.

Procedure₂ (Vote for Center node) in node_A

Input: $h_A^x, h_A^y, ID_A, h_C^x, h_C^y$, here C is a ideal local center

Output: $center_A$

```

delay(f(√((h_A^x - h_C^x)^2 + (h_A^y - h_C^y)^2)));
SEND:
if( receive(msg(center_A) from the nodes in this area)
  center_A = self;
  while(send(msg(center_A)) == collision)
    delay(f(√((h_A^x - h_C^x)^2 + (h_A^y - h_C^y)^2) + random(ID_A)));
    Goto SEND;
  endwhile
Endif

```

In our simulation, this simple voting algorithm works well, even in some exceptions like the one shown in Figure.1, where the voting mechanism produces more than one center node for a single local area. In the local area circled in red, three local center nodes are produced because the distances between some nodes are more than one hop.

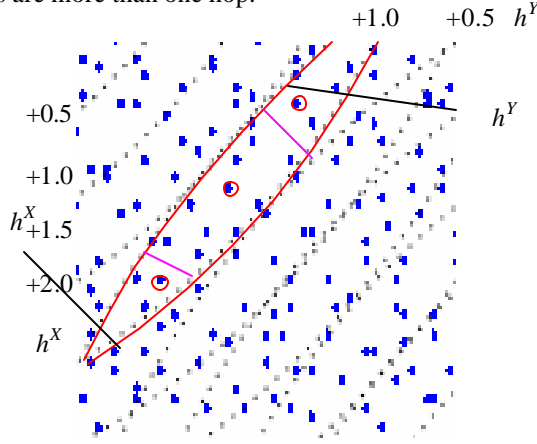


Figure 1. An exception for center node generation in a 2500 node WSN
The nodes with red circles are selected as center nodes for one area local area, whose diameter is >1hop.

Step 3.2: After the generation of at least one center node for each local area each center node will send a request to its neighbor nodes that are within *one* (or *k*) hop(s) to send back their hop coordinate from bootstrap node *X* or *Y*.

The total cost for this step is as follows: computational cost of $O(1)$, communication cost of $O(|N_A|)$, memory cost of $O(|N_A|)$ for center nodes and $O(1)$ for nodes that are not center nodes, and $O(n)$ for the whole network.

Step 4) : Apply MDS algorithm to calculate the local map in each center node.

Step 4.1: Similarly to MDS-MAP(P), each center node computes shortest paths between all pairs of nodes one (*k*) hop(s) to that node, using Dijkstra's algorithm or other similar algorithms.

Step 4.2: We then apply MDS to the $(|N_A|+1) \times (|N_A|+1)$ shortest path matrix (here $|N_A|$ is the number of nodes that can be reached by center node *A* in one (*k*) hop(s)) and retain the first two (or three) largest eigenvalues and eigenvectors to

construct a 2-D (or 3-D) local map. This step is the same as 2.b in MDS-MAP(P).

The total cost for this step is a computational cost of $O(|N_A|^3 n)$ and a memory cost of $O(|N_A|^2)$ per center node, with no communication cost in this step.

Step 5) : Calculate transformation matrix and transmit it to neighboring center nodes.

Step 5.1: We next find the set of neighboring center node for each center node *A*.

For this process, we need a bootstrap node. Node *Y* will be used as the bootstrap location for the center node of a given area, which we will refer to as node *A*, to run procedure₃.

Procedure₃ Find neighboring center nodes for node_A

Input: None,

Output: neighbor center set $C_A \subseteq N_A$

```

Send out an ASK message to ask for possible
neighbor center nodes within one (or k) hop(s) of
node A;
Receive the reply messages from neighbor centers;
Return neighbor center node set C_A;

```

Step 5.2: After we find the set of neighboring center nodes, we compute a transformation matrix for each node *B* in the neighbor center set C_A of center node *A*.

Suppose that there are two sets of neighbor nodes, N_A and N_B , that are within *one* (or *k*) hop(s) of the center nodes *A* and *B*, respectively. Their intersection is $I = N_A \cap N_B$, and we use

matrix $I_A = \{ \dots, (x_i, y_i), \dots \}$ (here (x_i, y_i) are the coordinates of one node in the map generated by node *A*) to represent the coordinates of nodes in the *I* generated by node *A*, and similarly $I_B = \{ \dots, (x'_i, y'_i), \dots \}$ for node *B*. We can then compute a transformation matrix *T* such that it minimizes $\text{sqrt}(\sum((x_i - x'_i)^2 + (y_i - y'_i)^2))$, where $(x_i, y_i) \in I_A$, $(x'_i, y'_i) \in I_B$, $(x''_i, y''_i) \in I_B \times T$. The procedure is as follows:

Procedure₄ Compute transformation matrix *T* in node_A

Input: neighbor center node set

Output: None

```

for each node B in
  request N_B from B;

```

$I = N_A \cap N_B$;

generate I_A, I_B ;

```

compute transformation matrix T such that
sqrt(∑((x_i - x'_i)^2 + (y_i - y'_i)^2)) is minimized
(here (x_i, y_i) ∈ I_A, (x'_i, y'_i) ∈ I_B × T);

```

```

send matrix T to node B;

```

```

endfor

```

If a center node receives a transformation matrix *T*, then it will first check whether it is already transformed or not; if so, the center node will drop such a message, and if not, it will

apply procedure5. This will allow the center node to compute the local map, which it will then send to its $|N_A|$ neighbor nodes. At the same time, the center node will find its neighbor center node set using procedure3, and will compute transformation matrix T for each center node in the set using procedure4.

Total cost for this step: computational cost of $O(|N_A|)$, memory cost of $O(|N_A|)$ for center nodes and $O(1)$ for nodes that are not center nodes, and communication cost of $O(1)$ for each node or $O(n)$ for whole network.

Procedure₅ (Receive T in node_A)

Input: matrix T from neighbor center node, from MAC layer

Output: None

If this node has been transformed

Drop(msg(T));

Exit;

set *self* as transformed;

transform the local map with T ;

send back to $|N_A|$ neighbor nodes.

$C_A = \text{procedure}_3()$;

for each node B in C_A

compute T with procedure₄() for node B ;

send T to node B ;

IV. SIMULATION RESULTS

A. Simulation Configuration

We implemented our localization algorithm as a routing agent and our bootstrap node candidate program as a protocol agent in NS-2 version 2.29 [10] with 802.15.4 MAC layer [13] and CMU wireless [14] extensions. The configuration used for NS-2 is RF range = 15 meters, propagation = TwoRayGround, antenna = Omni Antenna.

In our experiments, we used uniform placement— n nodes are placed on a grid with $\pm 0.5r$ randomized placement error. Here r is the width of a small square in the grid. We constructed a total of 60 placements with $n = 36, 100, 225, 400, 625, 900, 1600, 2025,$ and 2500 , and with $r = 2, 4, 5, 8, 10,$ and 12 meters, respectively. The reason we use uniform placement with $\pm 0.5r$ error is that usually such placement produces both node holes and islands in one placement, as demonstrated in Figure.2.

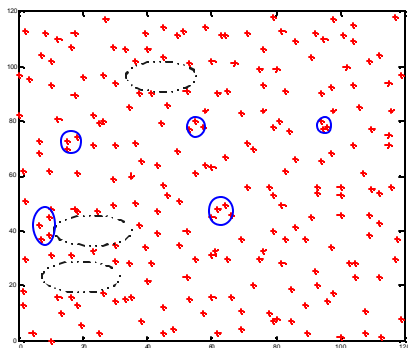


Figure 2. A typical placement for simulation Constructed with $n=400, r=4$. Black dotted ovals are holes and small blue circles are islands.

The MDS-MAP series of algorithms we used for comparing our results are based on [15] and [16]. We ran our localization algorithm in NS-2 and used MDS-MAP series algorithms running in Matlab as in [15-16] under the same placements as mentioned before. We didn't include the MDS-MAP algorithms' refine steps introduced in [15] and [16], but note that similar steps can be added to both our algorithm and the MDS-MAP/MDS-MAP(P) algorithms.

B. Simulation Result

In order to evaluate the accuracy of our algorithm in different size and different density WSNs, we ran our algorithm and compared the MDS-MAP series algorithms in the 60 placements described in section 4.1.

We compute the overall average accuracy corresponding to the various grid sizes (r) tested. These results are shown in Figure. 3(a-c). We can see from Figure. 6(a) that when $r \geq 10$, the error in the location generated by our algorithm is similar to that of the comparison algorithms, and that when $r \leq 2$, the improvement in the accuracy from our algorithm is not as same as when r has greater values. Since the application of the proposed algorithm was shown above to be poor when $r \leq 2$ and $n \leq 100$, as shown in Figure. 6(b), we calculated overall results, shown in Figure. 6(c), without this segment. Figure 6(c) shows that our algorithm improves the accuracy of localization about 45% and 36% respectively compared with the MDS-MAP and MDS-MAP(P) algorithms using only hop-counting, when $2 \leq r < 10$; these figures reduce to about 34% and 26% respectively compared with the MDS-MAP and MDS-MAP(P) algorithms, when $2 \leq r \leq 12$.

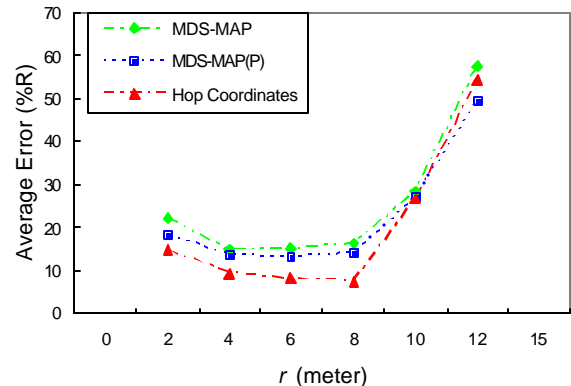


Figure.3(a). Comparison of overall average accuracy of localization. ($n=36-2500$)

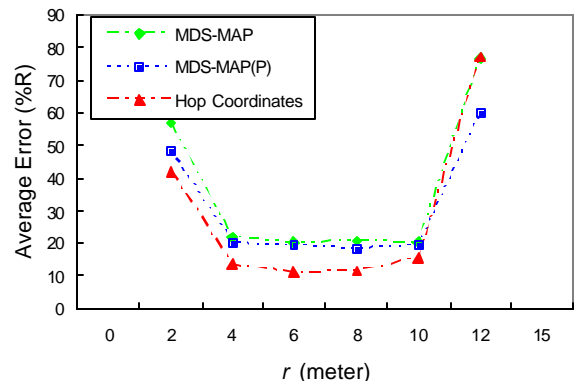


Figure. 3(b). Comparison of overall average accuracy of localization. ($n=36-100$)

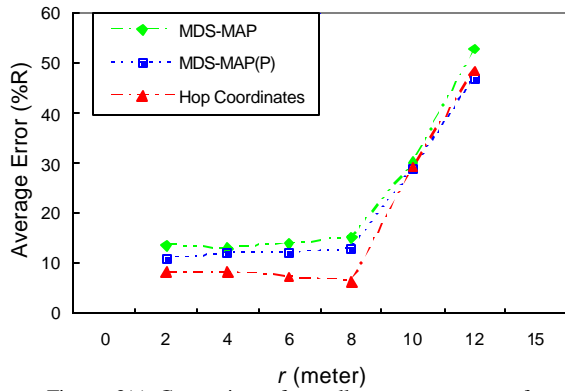


Figure 3(c). Comparison of overall average accuracy of localization. ($n=225-2500$)

V. SUMMARY AND CONCLUSION

This paper proposes a Geographic Distributed Localization (GDL) algorithm based on hop-counting. In addition to calculating hop numbers from bootstrap nodes, our method stores local network connections for a node by generating special “hop coordinates” that augment the hop number with additional information. Using a simple voting mechanism, our method generates a center node geographically distributed for each local area, each of which then creates a local relative map for its area. By computing and transmitting transformation matrix between adjacent center nodes, our algorithm avoids the merge process required by MDS-MAP(P). Extensive simulation shows that our algorithm improves the accuracy of localization by about 45% and 36% respectively compared with the MDS-MAP and MDS-MAP(P) algorithms. It also has a lower communication cost $O(n)$ for whole network, and has a low constant memory cost per node. Our simulation shows that the algorithm is effective over a wide range of cases, although in two specific cases, namely (1) when n is small and all nodes are close with each other (r is small) and (2) when all nodes are far away from each other (r is large), the proposed algorithm may not be helpful.

For future research, as more and more commercial sensor networks provide RSSI, it is worth noting that it is possible for our algorithm to be combined with RSSI in order to improve the accuracy of localization further. Also, while research has shown that the MDS method is suitable for anchor-free localization in WSNs, our algorithm can also be adapted to other emerging methods such as SDP (Semi-Definite Programming) [6].

REFERENCES

[1] I.F. Akyildiz, W. Su, Y. Sankarasubramaniam, E. Cayirci, “A Survey on Sensor Networks”, IEEE Communication Magazine, vol. 40, no. 8, pp. 102–114, Aug. 2002.

[2] M.A.M. Vieira, C.N. Coelho Jr., D.C. da Silva Jr., J.M. da Mata, “Survey on wireless sensor network devices”, Proceedings of IEEE conference on Emerging Technologies and Factory Automation, 2003 (ETFA '03), pp.:537 - 544 vol.1, Sept. 2003.

[3] D. Culler, P. Dutta, C. T. Ee, R. Fonseca, et al. “Towards a Sensor Network Architecture: Lowering the Waistline”, ACM SenSys 2005, Nov. 2005.

[4] T. He, C. Huang, B. Blum, J. Stankovic, and T. Abdelzaher. “Range-Free Localization Schemes in Large Scale Sensor Networks”. In Proc. of Mobile Computing and Networking (MobiCom2003), 2003.

[5] M. Maróti, P. Völgys, S. Dóra, B. Kusý, A. Nádas, Á. Lédeczi, G. Balogh, K. Molnár, “Radio interferometric geolocation”, In Proc. of the 3rd Intl. Conf. on Embedded networked sensor systems (SenSys'05), Nov., 2005.

[6] A. M.C. So, Y. Ye, “Theory of semidefinite programming for sensor network localization”, Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms(SODA), P405-414, 2005.

[7] R. Bischof, R. Wattenhofer, “Analyzing Connectivity-based, Multi-hop Ad hoc Positioning”, In Proc. of the IEEE Intl. Conf. on Pervasive Computing and Communications (PerCom), 2004.

[8] G. Zhou, T. He, S. Krishnamurthy, and J. A. Stankovic, “Impact of radio irregularity on wireless sensor networks”, Proceedings of the 2nd international conference on Mobile systems, applications, and services (MobiSys '04), pages 125–138, 2004.

[9] J.N. Ash and L.C. Potter, “Sensor network localization via received signal strength measurements with directional antennas,” in Proc. 2004 Allerton Conf. Communication, Control, and Computing, pp. 1861–1870. Sep 2004.

[10] S. McCanne, S. Floyd, “NS-2 Network Simulator”, <http://www.isi.edu/nsnam/ns/>.

[11] A. Savvides, C. Han, M. Srivastava, “Dynamic fine-grained localization in ad-hoc networks of sensors”, In Proceedings of ACM MOBICOM 2001, P.166–179.

[12] W. Du, L. Fang, P. Ning, “LAD: Localization Anomaly Detection for Wireless Sensor Networks”, Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05), p.41.1, April, 2005.

[13] J. Zheng, et al., “802.15.4 extension to NS-2”, <http://www-ee.cuny.edu/zheng/pub>.

[14] The CMU MONARCH Group, “Wireless and Mobility Extensions to ns-2”, <http://www.monarch.cs.cmu.edu/cmu-ns.html>.

[15] Y. Shang, W. Ruml, Y. Zhang, and M. Fromherz. “Localization from Mere Connectivity”, In Proc. of Intl. Symp. on Mobile Ad Hoc Networking and Computing (MobiHoc), 2003.

[16] Y. Shang and W. Ruml, “Improved MDS-based localization,” in IEEE Proc. Infocom'04, pp. 2640-2651, Mar. 2004.

[17] J. Bruck, J. Gao and A. Jiang “Localization and Routing in Sensor Networks by Local Angle Information”, Proc. the 6th ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc'05), May 2005.

[18] N. Priyantha, A. Chakabarty, H. Balakrishnan, “The Cricket Location-support System”, in Proceedings of MobiCom, 2000.

[19] S. Capkun, M. Hamdi, J. Hubaux, “GPS-free positioning in mobile ad-hoc networks”, In Proceedings of Hawaii International Conference on System Sciences (HICSS'01). P.3481–3490, 2001.

[20] Y. Xu, J. Ford, F. S. Makedon, “A Variation on Hop-counting for Geographic Routing”, IEEE Workshop on Embedded Networked Sensors (EmNetS-III), Boston, MA, 2006.