The FU Berlin Parallel Lateration-Algorithm Simulation and Visualization Engine

Heiko Will
Freie Universität Berlin
AG Computer Systems & Telematics
Berlin, Germany
Email: hwill@inf.fu-berlin.de

Thomas Hillebrandt
Freie Universität Berlin
AG Computer Systems & Telematics
Berlin, Germany
Email: hillebra@inf.fu-berlin.de

Marcel Kyas
Freie Universität Berlin
AG Computer Systems & Telematics
Berlin, Germany
Email: marcel.kyas@fu-berlin.de

***Abstract*—We introduce a simulation engine to visually evaluate and compare distance based lateration algorithms and deployments called the *FU Berlin Parallel Lateration-Algorithm Simulation and Visualization Engine (LS^2)*. Our engine simulates a scenario which consists of given anchor positions and evaluates all positions of a playing field in parallel, instead of only randomly selected positions. At the end of a simulation run, the user is able to judge the strengths and weaknesses of the algorithm in a picture that displays the *spatial position error distribution,* a representation of positions in a plane with their errors. Understanding spacial distribution of error is especially valuable, because we observed that it depends on the placement of anchor nodes. This enables the developer to optimize his algorithm or aid him in selecting an algorithm for his application.**

The simulator's design separates the simulation engine, the lateration algorithms, and the error models. The simulator can be easily extended with additional lateration algorithms and error models. The engine is written in GNU C99 and uses the SSE or AVX vector extensions of modern microprocessors. Thus, it is able to scale fully to all cores. Beside extendability, the main focus is set on execution speed.

I. Introduction

A distance-based localization algorithm, or lateration algorithm, uses m known locations, subsequently called anchors, and m distances to these anchors to estimate the location of a target node. Many algorithms have been proposed for this task, e.g., trilateration, linear least-squares (LLS), non-linear least-squares (NLLS), and Adapted Multi-Lateration (AML). Also, there have been attempts at judging the quality of such algorithms, usually by simulating them. The quality of an algorithm is usually measured by the *position error*, i.e., the distance between the calculated and the actual position. Since the 1960's there have been various approaches to analyze and handle this error [1].

At first glance, the simulation of a localization algorithm is a straightforward task. Commonly, the algorithm is run a large number of times with preselected (or sometimes random) anchor positions, random target positions, and a random *distance error,* an error added to the input distance measurements, computed from some error model. The result is a statistical evaluation of the accumulated position errors. At second glance, this approach has several flaws. For most distance based algorithms, the position error is not only a result of the errors of the distance measurements between the node and the anchors. We claim that the biggest contribution to the position error is a result of the node's position relative to the position of the anchors. For example, the position of the node relative to the triangle formed by the three anchors is much more important than the distance error for the simple trilateration algorithm. For the same distance error, the position error is much lower if the target node is located near the triangle's center of mass than if it is located far outside of the triangle.

This observation indicates, that the distribution of the position error depends on the position of the target node. Consequently, we define the *spatial distribution of errors* as a mapping from positions to error distributions. This spatial distribution appears to be much more interesting than the statistical distribution of errors at all locations.

Assuming that the spatial distribution of position errors is not related to a statistical distribution but to geometric shapes induced by the anchor positions, it appears to be very likely that most real world scenarios get much better results if the spatial distribution is taken into account in the placement of anchors. For example, assume an algorithm with an average position error of E, where one continuous area has the error 0 and another one $2E$. Then the best solution is to plan the network in a way that mobile nodes stay in the 0-error-shape. To do this, one has to gather a view of the spatial error distribution, i.e., every possible position on a given playing field has to be simulated thousands of times to differentiate between outliers in distance errors and position errors. Depending on the complexity of the algorithm and the size of the discrete playing field, the main loop of a simulation engine using the simulated algorithm has to be executed billions of times. Even on modern CPUs this can be a very time consuming task.

The contribution of this paper is twofold. First, we introduce the LS^2 simulation engine, including a library with vectorized implementations of the most common lateration algorithms and error models. The simulation engine can easily be extended with other algorithms and error models. The results of the simulation runs can be interpreted visually as two-dimensional images. Second, we present a first overview of the spatial distribution of position errors for four very common lateration algorithms and discuss their consequences for real

world applications.

The paper is structured as follows. In Section II we introduce the simulation engine and discuss its implementation. In Section III we evaluate the performance of the simulation engine and present benchmark results. In Section IV we addresses the spatial error distributions of four common lateration algorithms and their consequences for real world applications. Related work is discussed in Section V and in Section VI we present our conclusion and give an outlook to further research and improvements.

II. The Simulation Engine

The basic principle of LS^2 is that m positions of fixed anchors are placed on a user defined playing field and for every possible discrete location on the playing field the position of the node is calculated multiple times with a user provided algorithm and a user provided error model. At the end of a simulation run the simulator provides two sets of data. The first set consists of the average position error for every location and the second set of the highest position error for every location.

A. The Simulation Core

The LS^2 core is a loop which iterates through all node locations of a playing field of 1000×1000 discrete positions. For each location, the distance to all anchors is calculated and an error conforming to the selected error model is added. The anchor positions together with these erroneous distances are passed to the selected algorithm to estimate a position. Then, the error between the node location and the algorithm's result is accumulated. A simple dispatcher calls multiple instances of the core as threads with a subset of the positions to be calculated to make use of all cores of the used computer. Due to the independence of the calculations it should scale almost linearly with the number of CPU cores until the memory bandwidth limit of the system is reached. For a single simulation run, each location involves 1000 runs with different distance errors. As a consequence, the algorithm is called one billion times, in addition to one billion calls of the error model per anchor. Each call of the error model function incurs the generation of at least one pseudo-random number. To handle this massive amount of calculations the whole core and most of the provided algorithms are optimized for maximum performance and parallel computation.

To make use of parallelization beyond the thread level, the whole core utilizes the vector units of modern CPUs [2]. The core provides two vector models. The first one requires a SSE2 unit that should be present on every 64-bit CPU and is capable of calculating 4 single precision floating point operations simultaneously. The second option requires a AVX unit which is present in state-of-the-art Intel Sandy Bridge and AMD Bulldozer architectures and is capable of calculating 8 floating point operations at once.

For best performance, all algorithms and error models should also be optimized to use vector pipelines. Therefore, we are providing a number of utility functions with the simulation core. These utilities include some basic geometric functions to use in algorithms and a parallel random number generator to use in the error models. The random number generator is based on a linear congruential generator [3] and also uses vector pipelines through the whole generation process.

To minimize the overhead of function calls during the simulation no function calls at all are made by the core and all provided algorithms and error models. To achieve this goal and still provide readable C code, all functions are inlined and included in units of C-files. To build the LS^2 engine, GCC version 4.6 or later with link time optimization [4] (LTO) is required. The C dialect is GNU C99 which enables us to declare all variables as late as possible to minimize the number of cache misses.

The simulator can be built in two versions with the included makefile. The first version is a command line tool which includes only one user selectable algorithm and only one error model. This version provides the highest simulation speed and could easily be scripted to generate simulation results for a large number of simulations, e.g. for moving anchors, at a batch. The second version builds to a dynamic library which includes all algorithms and all error models but cannot totally eliminate function calls. This library can easily be integrated into other applications like graphical user interfaces (GUI). With the LS^2 engine we provide such a GUI application written in Java which directly calls the shared library with a remarkable low time overhead.

The result of a simulation run is written to disk as a single file consisting of the two data sets. A very basic image header (TGA) [5] is put in front of the data, so the results could directly be viewed in most image viewers. For further processing of the data the image header could simply be skipped. All visualizations in this paper are directly generated by the simulator.

B. Provided Algorithms

The current release provides four different lateration algorithms. For sake of analysis and benchmarking, we present a short overview of four common algorithms. A fifth algorithm called "const" always returns a fixed position on the playing field and does not need any computation time. If used without error model, this algorithm is used to benchmark the core of the simulator itself.

1) Trilateration: Trilateration is a method to determine the location of an object based on range measurements to anchor nodes at known locations. At least three distances to non-collinear anchors are required to calculate the coordinates of the target on an Eucledian plane. The problem can be expressed as finding the intersection of three circles, that is, finding the solutions of the following system of quadratic equations:

$$\begin{aligned}(x-x_1)^2+(y-y_1)^2 &= {r_1}^2\\ (x-x_2)^2+(y-y_2)^2 &= {r_2}^2\\ (x-x_3)^2+(y-y_3)^2 &= {r_3}^2\end{aligned}$$

where (x, y) is the coordinate of the target, $(x_i, y_i), i = 1, 2, 3$ are the coordinates of the anchors and r_i the corresponding

distance. Because the known nodes' coordinates and distances normally include measurement errors, the system has no unique solution in general. However, a computationally inexpensive closed-form solution exists and can be found in [6] that correctly calculates the target position if it is unique.

2) Nonlinear Least Squares Multilateration: Multilateration works much like Trilateration but with the advantage that it can incorporate more than three anchor nodes in the position estimation process. Given m anchor nodes with fixed positions at $b_i = (x_i, y_i)$ with $i = 1, 2, \ldots, m$ and possibly noisy range measurements r_i from these nodes to a non-anchor node located at $u = (x, y)$, multilateration finds the most likely position of the unknown node, denoted by $\hat{u}$. Again, from the given information we can write a system of equations as:

$$\begin{aligned} (x - x_1)^2 + (y - y_1)^2 &= {r_1}^2 \\ (x - x_2)^2 + (y - y_2)^2 &= {r_2}^2 \\ &\vdots \\ (x - x_m)^2 + (y - y_m)^2 &= {r_m}^2 \end{aligned} \tag{1}$$

where x and y are the only unknowns in the above nonlinear equation system. This problem is usually solved by using a *least squares* method, that is, minimizing the sum of the squared residuals between the observed ranges r_i and the estimated distance $||u - b_i||$:

$$\hat{u} = \underset{u}{\operatorname{argmin}} \sum_{i=1}^{m} \big(||u - b_i|| - r_i \big)^2 \tag{2}$$

The minimization problem can be solved by using any of the Newton type optimization algorithms [7]. These start from an initial guess at the solution and then do a number of iterations. Each iteration gradually improves the estimated position until a local minimum of the objective function in Eq. (2) is found.

3) Linear Least Squares Multilateration: The *nonlinear least squares* problem can be linearized by subtracting one of the equations given in Eq. (1) from the remaining $m - 1$ equations. The linear system can be expressed as

$$Au = b$$

where

$$A = \begin{bmatrix} x_1 - x_m & y_1 - y_m \\ x_1 - x_m & y_1 - y_m \\ \vdots & \vdots \\ x_{m-1} - x_m & y_{m-1} - y_m \end{bmatrix}$$

and

$$b = \frac{1}{2} \begin{bmatrix} x_1^2 - x_m^2 + y_1^2 - y_m^2 + r_m^2 - r_1^2 \\ x_2^2 - x_m^2 + y_2^2 - y_m^2 + r_m^2 - r_2^2 \\ \vdots \\ x_{m-1}^2 - x_m^2 + y_{m-1}^2 - y_m^2 + r_m^2 - r_{m-1}^2 \end{bmatrix}$$

This linear regression problem is known as *linear least squares* and can be solved by the closed form solution shown in Eq. (3).

$$\hat{u} = \left(A^T A \right)^{-1} A^T b \tag{3}$$

To avoid trapping into local minimum when using NLLS it is favorable to use the LLS solution as a starting point for the NLLS optimization process because the linear estimation lies beneath the global minimum [8].

4) Adapted Multi-Lateration: Similar to multilateration, Adapted Multi-Lateration [9] tries to estimate the position of an unlocalized node using circle intersections. In addition, AML aims to reduce the computational overhead involved with matrix calculations in multilateration, resulting in a runtime in $\mathcal{O}(n^3)$, using a heuristic approach to a runtime in $\mathcal{O}(n^2)$ [9]. AML consists of three steps: *intersection and elimination, first estimation* and *refinement*. At the first step two intersecting circles are arbitrarily chosen. These circles may intersect at one or two points. If there is more than one point, the point with the larger distance to the third anchor is eliminated.

At the *first estimation step* the previously computed intersection point is moved to the middle of the line connecting it with the closest point of the third anchor's circle. This is done to compensate the errors introduced by range measurements. The calculation is done using the resemblance of triangles.

At the last step the position can be further refined. Therefore, the anchors that were not used in the previous steps are added to the position estimation process with the same principle utilized in the second step.

C. Error Models

The current release provides three error models. Beside a simple model based on a uniform distribution where the maximum error is the only variable there are two Gaussian distributed models. The first one is configurable by mean and variance. The second one adds a configurable possibility for a non-line-of-sight (NLOS) error. If the error is considered as NLOS it is multiplied with a configurable constant. The modeling of NLOS errors is described and discussed by Heidari and Pahlavan [10].

III. Performance Analysis and Discussion

To analyze the performance of the simulator we have run several test simulations. We varied the number of anchors, the number of used CPU cores and the used algorithms. For a general rating of the results we have also run all tests with a performance optimized Java library we used for automatic testing of our C implementations. All benchmarks were run on an Intel XEON E31245 quad core which possesses four physical cores and eight logical cores due to hyper-threading. The core speed was 3.3 GHz. The playing field always had a size of 1000×1000 discrete positions and every position was calculated 1000 times. So every algorithm was called one billion times and due to the usage of the simple error model one billion random numbers were needed per anchor. Every simulation run was conducted ten times and the average runtime was taken.

Table I shows the relation between the number of threads and the runtime of the simulated algorithms. Except NLLS

TABLE I
CPU BENCHMARK

Algorithm	1 Thread	2 Threads	4 Threads	8 Threads
Const	0.77 s (100%)	0.40 s (104%)	0.23 s (119%)	0.18 s (187%)
Trilateration	6.23 s (100%)	3.25 s (104%)	1.69 s (109%)	1.48 s (190%)
LLS	17.41 s (100%)	9.09 s (104%)	4,80 s (110%)	3.77 s (173%)
AML	83.23 s (100%)	42.97 s (103%)	22.68 s (109%)	17.65 s (170%)
NLLS	230.58 s (100%)	129.99 s (113%)	71.59 s (124%)	37.5 s (187%)

The table shows the ability of the simulation engine to scale with the number of used CPU cores. All benchmarks were run on a four core CPU with hyper-threading enabled. In brackets we show the overall consumed CPU time as a ratio to the one thread version.

TABLE II
THE CORRELATION BETWEEN EXECUTION TIME AND THE NUMBER OF ANCHORS.

Algorithm	3 Anchors	6 Anchors	9 Anchors
Trilateration	1.48 s	1.83 s (124%)	2.35 s (159%)
LLS	3.77 s	5.79 s (154%)	7.89 s (209%)
AML	17.65 s	37.75 s (213%)	59.07 s (334%)
NLLS	56.90 s	87.62 s (154%)	114.11 s (201%)

Percentages represent overhead to 3 anchors.

TABLE III
COMPARISON BETWEEN JAVA AND C IMPLEMENTATION

Algorithm	Runtime Java	Runtime SSE	Speedup
Trilateration	4.577 s	0.277 s	16.5
LLS	80.688 s	0.685 s	117.8
NLLS	449.257 s	11.971 s	37.5
AML	10.610 s	6.027 s	1.8

all other algorithms are scaling nearly proportional until the number of threads reaches the number of CPU cores. The runtime of const consists of the big simulation loop and the write-back of the result file. Because no expensive calculations are done in the loop and the write-back time remains constant the scaling to more than two threads stays slightly behind the scaling of the other ones. NLLS does not scale as good as the other ones because the dispatching to threads is done statically and not related to the resulting amount of calculations. The iteration of NLLS terminates related to the error which leads to the observation that a thread which computes a region of the playing field with less error has to compute much more than one working on a region with higher error. The scalability to eight threads is very poor because only the CPU which does the interrupt handling for the operating system will interrupt a simulation thread and make any use of the faster task switching due to hyper-threading. We analyzed with a profiler that this observation is not caused by memory bandwidth limitations.

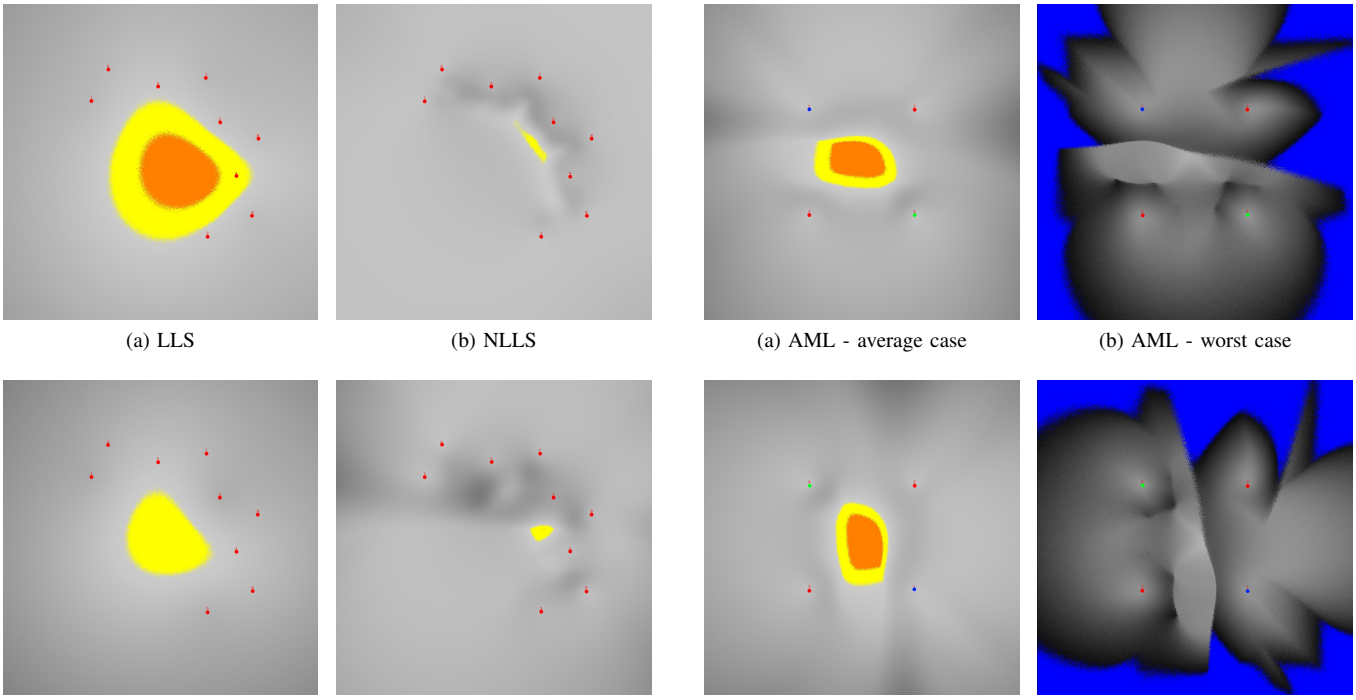
In Table II we show the correlation between execution time and the number of anchors. As trilateration only uses the first three of the given anchors the time penalty for more anchors is only resulting in the error model which is always applied to all calculated ranges regardless of the used algorithm. The overhead which would be produced by eliminating this penalty for trilateration would result in much bigger overhead for all other algorithms which are all capable of using all of the anchors. Also remarkable is that the implementation of AML performs really poor compared to LLS which is not this more complex than AML. The reason for that behavior is that AML could not really be vectorized which means that in case of the SSE implementation every run has to calculate 4 times more operations than the other algorithms. In section II-B4 we bounded AML with $\mathcal{O}(n^2)$ but the benchmark shows that the average case could be bounded with $\mathcal{O}(n)$.

Table III compares the runtime of an optimized Java reference implementation of the algorithm and the C implementation of the simulator, optimized for cache line usage, vector units and branch prediction hits. The benchmark shows that the improvement of the optimized implementation is massive. Only the AML algorithm which uses nearly the same implementation in C and in Java has a small speedup of 1.8. The other algorithms gain speedups up to 117.8 and are showing the very heterogeneous optimization capabilities of the Java environment.

IV. VALIDATION AND EXPERIMENTS

To validate the functionality of the simulation engine we ran some simple experiments and discuss them here. All images are taken directly out of the simulator, only the red anchor dots have been resized for a better visibility in this paper. All images are showing the spatial error distribution for each experiment. The bigger the amount of the error gets the darker the position in the image is colored. If the error is below the expected value of the distance error the location is colored yellow, or orange if the error is below 80% of the expected distance error. If the error is more than 500% of the expected distance error the location is colored blue to keep the contrast of the picture as low as needed. All color boundaries are freely configurable. The anchors are painted in red if not otherwise specially mentioned.

The boundary between lower position errors than the expected distance error and higher position errors than the expected distance error is very important for scenarios where the whole localization process is done recursively, e.g. in multi-hop scenarios. In those scenarios one can calculate the lateration with known anchor positions and recursively estimated distances or with recursively gathered positions and measured one-hop distances. The quality of those two approaches is directly derived from this observation. If every node which is part of the recursion is inside the yellow area then the recursion over the positions has a lower error, otherwise it's vice versa.



(a) LLS (b) NLLS (c) Trilateration (d) AML

Fig. 1. Spatial distribution of the position error with four different algorithms (average case).

(a) AML - average case (b) AML - worst case (c) AML - average case (d) AML - worst case

Fig. 2. Spatial error of AML with exchanged anchor positions

All simulation runs are done with 1000 iterations per position and a uniform error distribution with a maximum of 10% of the playing field length. The error is defined to be positive only because our real world anchor nodes only suffer positive distance error. The playing field is 1000×1000 units in size.

A. Comparing Algorithms

Figure 1 shows the spatial distribution of the average position error with four different algorithms. We used 9 anchors placed in the middle of the playing field for every algorithm whereas trilateration used the three anchors spanning the largest triangle area. The average position error for each algorithm is:

- LLS - 65 simulation units
- NLLS - 63 simulation units
- Trilateration - 83 simulation units
- AML - 75 simulation units

In a commonly used statistical evaluation trilateration would have been the worst performing algorithm followed by AML and LLS and NLLS on the top position. A look on the spatial distribution of the position error could lead to different conclusions. Looking at the spatial distribution one could easily see that there is a large region where trilateration performs much better than NLLS and an even larger region where LLS performs much better than all other algorithms. This observation could be very important for scenarios where one can choose from a number of anchors and the position of the mobile node is roughly known. This is the case in nearly all tracking applications.

For most indoor applications only the area inside the convex hull of the anchors is of interest because some anchors are normally mounted around the building to receive GPS signals. For such setups it is very important to place the anchors in a way that the area of interest is inside the colored region. In mobile ad-hoc networks (MANET) NLLS could be a better choice because it has a very steady spatial distribution.

In the simulated setup with many anchors AML only has its strength outside the convex hull. If it would be possible to get a mathematical description of the spatial distribution even a dynamic choice of the lateration algorithm is thinkable.

B. Analyzing an Algorithm

In Figure 2 you can see the results of two runs of the AML algorithm with exchanged anchor positions. In Fig. 2c and Fig. 2d the coordinates of the blue and the green anchor have been swapped. In Fig. 2a and Fig. 2c you can see the average position error and in Fig. 2b and Fig. 2d you can see the maximum position error. Obviously, AML is not independent of the order of the anchors passed to the algorithm. If we exchange the anchor coordinates the resulting image is diagonally mirrored. This behavior can be explained with the first step of the AML algorithm. Because our implementation arbitrarily picks the first intersecting circles as recommended in [9], AML performs differently with the same input. This fact can be easily seen with our simulator but not when comparing the average position error which is the difference between the estimated and true locations. The average position error of both

images is 81.21 simulation units. Thus, AML might be further improved by implementing a better strategy of choosing the initial anchors.

V. Related Work

The error distribution of lateration algorithms is a problem which is discussed for years now. Several research approaches have focused on the statistical distribution of the position error [11]–[14]. Some researchers discussed the relation between the number of anchors and the position error [15], [16].

Few approaches discuss the relationship between anchor placement and node position. Savvides et al. are presenting an evaluation of multi-hop localization and the influence of anchor position on the position error [17]. They are showing some graphics about the spatial error distribution, but not very fine grained and they are not providing a structured simulation environment.

Yang et al. are presenting a slightly similar approach to research the spatial error distribution. Instead of calculating every possible node position on the playing field and visualizing the error distribution, they are calculating one position on the playing field and visualizing the possibility for this node to be located on all other positions of the playing field [12].

In the science journal Navidi et al. are giving a very profound discussion about two of the main lateration approaches, trilateration and multilateration. They conclude with the finding that further research to the domain of anchor placement and about the influence of anchor position to the lateration result would be valuable [18].

For steady error distributions in some publications the Cramer-Rao Bound (CRB) is calculated instead of a simulation approach. Dulman et al. are showing in their paper that the CRB does not fit for the localization problem [19].

VI. Conclusion

We have presented LS^2 as a parallel lateration algorithm simulation engine with a focus on the evaluation of the spatial position error distribution. We showed that a visualization of the spatial position error distribution could lead to a different quality estimation of lateration algorithms than the evaluation of those algorithms with statistical approaches. Together with the simulation engine we provide an performance optimized library with common lateration algorithms. The whole software package is freely available under an open source license [20].

We suggest LS^2 as a general tool for the comparison and analysis of lateration algorithms.

Future work should address the integration of new algorithms and more sophisticated error models. A map based error model and the possibility to integrate a propagation model would also be a step forward. The random number generator should make use of the next processor generation hardware random number generators to improve performance and to enhance the random number distribution.

References

[1] J. Powers, "Range trilateration error analysis," *Aerospace and Electronic Systems, IEEE Transactions on*, no. 4-Suppl, pp. 572–585, 1966.

[2] A. J. Bik, *Software Vectorization Handbook, The: Applying Intel Multimedia Extensions for Maximum Performance*. Intel Press, 2004.

[3] H. Leeb and S. Wegenkittl, "Inversive and linear congruential pseudorandom number generators in empirical tests," *ACM Trans. Model. Comput. Simul.*, vol. 7, pp. 272–286, April 1997. [Online]. Available: http://doi.acm.org/10.1145/249204.249208

[4] The GNU project, "Link-time optimization in gcc: Requirements and high-level design," November 2005. [Online]. Available: http://gcc.gnu.org/projects/lto/lto.pdf

[5] Truevision, Inc., "Truevision tga file format specification version 2.0," 1991. [Online]. Available: http://www.dca.fee.unicamp.br/m̃artino/disciplinas/ea978/tgaffs.pdf

[6] R. Mardeni and S. N. Othman, "Node positioning in zigbee network using trilateration method based on the received signal strength indicator (rssi)," *European Journal of Scientific Research*, vol. 46, pp. 048–061, 2010.

[7] J. E. Dennis, Jr. and R. B. Schnabel, *Numerical Methods for Unconstrained Optimization and Nonlinear Equations (Classics in Applied Mathematics, 16)*. Soc for Industrial & Applied Math, 1996.

[8] Z. Li, W. Trappe, Y. Zhang, and B. Nath, "Robust statistical methods for securing wireless localization in sensor networks," in *IPSN 2005. Fourth International Symposium on Information Processing in Sensor Networks, 2005.* IEEE, 2005, pp. 91–98. [Online]. Available: http://dx.doi.org/10.1109/IPSN.2005.1440903

[9] G. S. Kuruoglu, M. Erol, and S. Oktug, "Localization in wireless sensor networks with range measurement errors," *Advanced International Conference on Telecommunications*, vol. 0, pp. 261–266, 2009.

[10] M. Heidari and K. Pahlavan, "A new statistical model for the behavior of ranging errors in toa-based indoor localization," in *Wireless Communications and Networking Conference, 2007. WCNC 2007. IEEE.* IEEE, 2007, pp. 2564–2569.

[11] J. Hightower and G. Borriello, "Location systems for ubiquitous computing," *Computer*, vol. 34, pp. 57–66, August 2001. [Online]. Available: http://dx.doi.org/10.1109/2.940014

[12] Z. Yang and Y. Liu, "Quality of Trilateration: Confidence Based Iterative Localization," *icdcs*, vol. 0, pp. 446–453, 2008. [Online]. Available: http://dx.doi.org/10.1109/ICDCS.2008.59

[13] I. Guvenc, C. Chong, and F. Watanabe, "Analysis of a linear least-squares localization technique in los and nlos environments," in *Vehicular Technology Conference, 2007. VTC2007-Spring. IEEE 65th*. IEEE, 2007, pp. 1886–1890.

[14] S. Slijepcevic, S. Megerian, and M. Potkonjak, "Characterization of location error in wireless sensor networks: analysis and applications," in *Proceedings of the 2nd international conference on Information processing in sensor networks*, ser. IPSN'03. Berlin, Heidelberg: Springer-Verlag, 2003, pp. 593–608. [Online]. Available: http://dl.acm.org/citation.cfm?id=1765991.1766032

[15] J. N. Ash and R. L. Moses, "On optimal anchor node placement in sensor localization by optimization of subspace principal angles," in *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing, ICASSP 2008, March 30 - April 4, 2008, Caesars Palace, Las Vegas, Nevada, USA*. IEEE, 2008, pp. 2289–2292.

[16] S. Yi, S. Hongchi, and A. Ahmed, "Performance study of localization methods for ad-hoc sensor networks," in *2004 IEEE International Conference on Mobile Ad-hoc and Sensor Systems*, 2004, pp. 184 – 193.

[17] A. Savvides, W. Garber, S. Adlakha, R. Moses, and M. Srivastava, "On the error characteristics of multihop node localization in ad-hoc sensor networks," in *Information Processing in Sensor Networks*, ser. Lecture Notes in Computer Science, F. Zhao and L. Guibas, Eds. Springer Berlin / Heidelberg, 2003, vol. 2634, pp. 555–555.

[18] W. Navidi, W. S. Murphy, and W. Hereman, "Statistical methods in surveying by trilateration," *Computational Statistics & Data Analysis*, vol. 27, no. 2, pp. 209–227, April 1998. [Online]. Available: http://ideas.repec.org/a/eee/csdana/v27y1998i2p209-227.html

[19] S. Dulman, P. Havinga, A. Baggio, and K. Langendoen, "Revisiting the cramer-rao bound for localization algorithms," *4th IEEE/ACM DCOSS Work-in-progress paper*, 2008.

[20] "LS^2 - lateration simulator." [Online]. Available: http://inf.fu-berlin.de/groups/ag-tech/Projects/ls2