Beyond Theory: Deployment of a Real-World Localization Application as Low Power WSN

Clemens Mhlberger, Marcel Baunach, Reiner Kolla
University of Wuerzburg, Germany

Beyond Theory: Development of a Real World Localization Application as Low Power WSN

Marcel Baunach, Reiner Kolla, Clemens Mühlberger
Department of Computer Engineering, University of Würzburg, Am Hubland, 97074 Würzbrug, Germany
Email: {baunach, kolla, muehlberger}@informatik.uni-wuerzburg.de
Telephone: +49 (0)931-888-6716, Fax: +49 (0)931-888-6702

***Abstract*—The real-world implementation of a just theoretically elaborated idea is sometimes cumbersome, often a couple of obstacles have to be overcome. That's likewise in the area of Wireless Sensor Networks (WSN), but complicated by some further restrictions, e.g. little memory or low power consumption. One well-known and often required application within WSN is the geographical localization of several sensor nodes. That's why this paper deals with some problems arising during the development of a WSN using the time difference of arrival (TDoA) of ultrasound and radio signals for positioning. Its focus is on handling of microcontroller difficulties like little memory, low computational power or low energy consumption as well as hardware driven failures like inaccurate measurements or node failures.**

***Index Terms*—real world implementation, WSN, TDoA localization, ramping, square root optimization**

I. Introduction

Lots of WSN applications are in need of knowledge about the precise positions of several objects. Because this is an elementary challenge, quite a number of (theoretical) solutions using different methods for this problem exist, like Active Bat [1], Cricket [2], [3], DOLPHIN [4], AHLoS [5] or SNOW BAT[6]. But their implementation is said easier than done, because some algorithms make excessive use of floating point (FP) numbers or complex (geometrical) functions like square root or arc tangent. Those are infeasible or not executable within an adequate amount of time by lots of microcontrollers. Some developed localization systems lack of fault tolerance or are not capable of real-time operation, which however is sometimes necessary to gain accurate positions.

This paper describes our own experiences gained during the development of a WSN based localization system and some of the lessons we learned. For system setup we used the sensor node platform SNOW^5[7], because wireless radio communication is yet available and supported as well as its great expandability to stack further sensor modules. Thus, this field report first introduces an idea for a WSN localization system using TDoA in section II and will then analyze physical aspects of the measurement procedure in section III. Afterwards one possible implementation for an ultra low power microcontroller as used in SNOW BAT will be worked out, keeping an eye on low computational power as well as on fault tolerance of the WSN (see section IV). A short summary closes this paper.

II. Hundred ways of localization

This section will introduce some characteristics for localizing systems (II-A). Next, a few approaches for distance measurement will be analyzed (II-B), a specific one will be selected for further examination (II-C).

A. Characteristics

Before examining some techniques for distance measurement, we will characterize localization systems by using the following parameters:

- The objects of the underlying system can be localized either *relative* to each other or *absolute* to a given reference point and coordinate system respectively.
- The process of localization can occur either *periodically* or *sporadically*, just when required.
- The *initiator* of the localizing process can either be the object to be located itself or (objects of) the surrounding environment.
- One can also distinguish between *active*, *passive* and *interactive* localization. The first means that an object determines its position autonomously within an environment which is passive in this regard. Whereas the second means that the position of an object is determined solely by the environment. Finally, *interactive* mode combines the two previous methods and thus requires an adequately equipped object and environment.
- The implemented algorithm can support only *two-dimensional* coordinates for easier calculations or – more generally – even *three-dimensional* ones.
- The estimation of distances and positions respectively can be fast enough to *track* mobile nodes up to a certain velocity or to just *locate* static ones.
- As mentioned in Bulusu [8], localization systems can also be determined by the coupling of the nodes which already know their positions, so-called *anchor nodes*. In a *tightly coupled* system the anchors are wired to a central unit whereas all nodes of the *loosely coupled* system use wireless communication.
- Also according to Bulusu [8], in a *centralized* structure a centric device controls measurement and calculates the different positions, unlike in *decentralized* systems. The first one requires central units with sufficient computational and managerial power, the latter causes considerable network traffic resulting in increased energy

consumption of the overall system for extra coordination. In *hierarchical* structures special units are privileged, that means more to control but also more to compute. Additionally, an *iterative* system needs a few runs for exhaustive localization.

B. Distance measurement methods

As mentioned in Beutel [9] and Tseng et al. [10], several techniques for distance measurement exist, each observing different physical quantities:

- *received signal strength (RSS)*
  This method tries to estimate the distance between two objects by measuring the strength of a received signal, as implemented for example by Hightower and companions [11] or at RADAR [12]. In accordance with *Coulomb's law* the field strength of an omni-directional electric field depends on the square of the distance from the field emitter. Indeed, the propagation of an electric field not only interferes heavily with the surrounding area but is also influenced by the antennas used or their design respectively. Thus an electric field spreads typically irregular in space, so it is hard to detect distances very precisely by RSS measurement (cf. Savvides et al. [5]).
- *angle of arrival (AoA)*
  It is also possible to determine one's position by trigonometrical calculations. Therefore you have to use hardware, which is able to detect the angle of a received signal. But this is the disadvantage at the same time, because sensing angles is somewhat difficult and the additionally required hardware is sometimes expensive. For example the Cricket Compass [2] needs five ultrasonic (US) receivers at each mobile node, a so called *compass*.
- *time of arrival (ToA)*
  Distance can be estimated also by measuring the time of arrival of an adequate signal. If for example two objects make a two-way handshake, the distance can be calculated by measuring the round-trip time of this signal at known signal speed. But you have to distinguish very well about the signal type to be used, because the slower the signal the longer takes the measurement and in contrast the faster the signal the more precise clocks for time measurement are essential. Especially the two-handshake method requires signal transceivers on each object.
- *time difference of arrival (TDoA)*
  Two signals traveling at different but known speeds are needed therefore. Distinctly diverging velocities make measurement of their time of arrival easier, that's why a combination of radio and ultrasound is very popular. In comparison to the two-way handshaking ToA using ultrasound, the devices of TDoA don't require a transceiver each and the slower ultrasound travels just one-way. Therefore such measurements can be managed by TDoA within less time and simply less ultrasonic hardware.

As RSS indication lacks accuracy and angles are hard to sense exactly for AoA, we face the choice between ToA and TDoA. Both methods are somewhat similar, but for the two-way handshaking ToA each node requires transceiver units. The measurement duration also mainly depends on the time of flight of the slowest signal used. So, if both methods would use ultrasound, TDoA could measure at ideal case almost twice as often as the two-way handshaking ToA (cf. Fig. 1). Thus we will examine TDoA in detail before we will develop an algorithm for localization by means of TDoA delivered distances.

C. TDoA – functionality and usage

As mentioned above, two signals traveling at different speeds are required. Radio and ultrasound are suitable, because radio travels at speed of light and is on dimensions faster than ultrasound. This makes the whole procedure comparable with a thunderstorm: you first see the flash and after a while you can hear the thunder. As denoted in Fig. 1, a sensor node m transmits a radio packet (*flash*) at time t_{RF_TX} to signalize a subsequently sent ultrasound (*thunder*) at time t_{US_TX}. Another sensor node s receiving the radio packet at time t_{RF_RX} activates its ultrasonic receiver. If s also detects an ultrasonic signal at time t_{US_RX}, the distance d can be calculated from additionally known velocities of ultrasound v_{US} and radio v_{RF} (cf. Tseng et al. [10]):

$$d = \big((t_{US\_RX} - t_{RF\_RX}) - (t_{US\_TX} - t_{RF\_TX})\big) \cdot \frac{v_{US} \cdot v_{RF}}{v_{RF} - v_{US}} \quad (1)$$

The sensor nodes must not only know the exact velocities of radio and ultrasound, but they also need to detect the precise point in time of transmission or reception respectively.

A few configurations for the realization of TDoA using ultrasound and radio within a WSN are conceivable. First, the WSN may only consist of static nodes. But if so, the positions of the nodes must be determined once and are valid forever. The other extreme will be a WSN made up of only mobile nodes. This configuration is much more complex and additional hardware like GPS modules for knowledge about the approximate position of some sensor nodes is essential if not only relative and inaccurate positions are desired. More interesting but quite simple realizable is a mixture of anchors, i.e. static nodes with known positions, and mobile ones which need to be localized.

In this context, the localization process can be initiated either by the static anchors like in Cricket [13] or by a

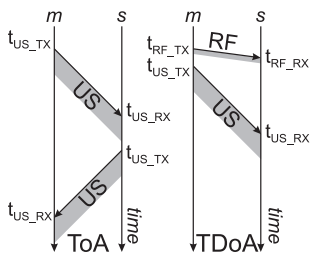

Fig. 1. Two-handshake ToA and TDoA in principle

mobile one itself. The latter is more energy-saving because the static nodes have to turn on their ultrasonic receivers not until they notice a radio signal from a mobile one initiating its localization. Otherwise they would have to send their signals continuously and periodically to avoid missing a mobile one but even if no mobile node is nearby.

We want to emphasize the wireless networking character as well as the distributed nature of the system to-be. That's why we prefer a loosely coupled and decentralized system in contrast to other existing systems like Active Bat [1]. Additionally, for a loosely coupled system, there are no cables to be laid and you are more independent during installation. Within a centralized system, the central unit with more computational power can make most of the calculations whereas a decentralized system would have to calculate everything within the network, but on the other hand seems to be more fault tolerant against single point of failure. A central unit could also lack scalability if the number of maximum connections is limited as with Bluetooth [14]. Next, we will have a detailed look at the TDoA distance measurement process.

III. TDoA measurement improved

The localization algorithm should calculate a concrete and accurate position for the mobile device, using only information like the coordinates and detected distances of sufficient static anchors. Because these coordinates are fixed and well-known only the measured time difference of arrival of radio and ultrasound need to be converted into a concrete distance. So, this section first presents one possible realization of a TDoA system (III-A) and derives therefrom an improved distance estimation algorithm (III-B). Next, the physical phenomena *ramping* will be examined (III-C) as well as the possibility of multichirping (III-D).

A. One realization of TDoA

As mentioned above, if a mobile node wants to get current position data, it sends a certain radio packet to start the localization process. Because radio signals in general range much further than ultrasound, a single hop message is sufficient and no routing is required. SNoW Bat calls such a message *chirp allocation vector (CAV)* which contains all data necessary, e.g.

- the ID of the mobile transmitter,
- its last known position pos_{old},
- its maximum speed v_{max} and
- a duration $\Delta t_{chirp} = t_{US\_TX} - t_{RF\_TX}$.

SNoW Bat expects the mobile node to *chirp*, i.e. to send an ultrasonic pulse, subsequent to and Δt_{chirp} after this CAV. Since a square wave signal is sufficient for stimulating the ultrasonic transmitter, for example within the MSP430x1xx MCU family [15], it may be created from time intervals, which are generated by the capture/compare block of either of the MCU's timers.

In contrast to radio, ultrasound does not emit omnidirectional but ideally as spherical sector (cf. Fig. 2), defined by its aperture angle θ (typically $\theta \approx 30°$) and its range d_{US},

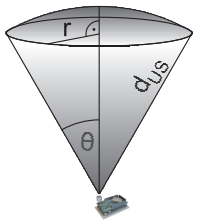

Fig. 2. Anatomy of an emitted ultrasonic signal

i.e. the length of its surface of revolution. Supposed, the maximum range $\max d_{US}$ of the ultrasonic transmitter used is about $10\ \mathrm{m}$, a radio signal would just need $\max \Delta t_{RF} = 33.3\ \mathrm{ns}$ to travel that distance. This is not only much too short to be detected by microcontrollers running at some MHz, but also the transmission time of a CAV sized $50\ \mathrm{B}$ will already take $\Delta t_{CAV} = 1.6\ \mathrm{ms}$ at an exemplary data rate of $250\ \mathrm{kbps}$. Hence the result of the distance measurement won't be falsified significantly, supposing the time difference of arrival of the radio signal as

$$\Delta t_{RF} = t_{RF\_RX} - t_{RF\_TX} = 0\ \mathrm{ms}. \tag{2}$$

All static nodes receiving a CAV must record the time of arrival of this radio packet at first and in addition activate their US receivers immediately, not to miss the following US chirp. But an anchor receiving a CAV could be out-of-range for the corresponding US signal or the chirp itself could be too weak for clear detection. Taking (2) into account, to save energy any anchor needs to activate its US receiver after CAV reception for only at most $\Delta t_{timeout} = \Delta t_{chirp} + \frac{\max d_{US}}{v_{US}}$. Seemingly, another advancement could be a prior test, whether the anchor is within the ultrasonic range of the CAV sending mobile node and thus has to activate its US receiver, or not. But keep in mind, this calculation does not only need detailed knowledge about the last known position pos_{old} and the maximum velocity v_{max} of the mobile device as well as the maximum range of the ultrasonic emitters, but its computation could also take longer than the intended time between radio signal and chirping, and thereby causing the real chirp to be missed.

However, if a static node receives a chirp after a foregoing CAV, it must also record the chirp's time of arrival and can shut down its ultrasonic receiver afterward for energy savings. The ultrasonic detection can be realized by an analog-to-digital converter or by a capture/compare unit, which will be more adequate and more energy-saving.

B. Distance estimation

Assuming the things specified above, the distance d between the chirping mobile node and the receiving anchor can be computed then solely by the static node using an improved version of (1) as follows:

$$d = (t_{US\_RX} - t_{RF\_RX} - \Delta t_{chirp}) \cdot v_{US} \tag{3}$$

In contrast, this equation contains no more division and thus saves CPU time. It demonstrates as well, the more precise

the time recording and the determination of v_{US} are, the more accurate is the calculated distance. That means, the interrupt latency at the nodes must be kept short and constant at most, especially to properly identify the different arrival times t_{US_RX} and t_{RF_RX}. The speed of ultrasound also has to be quantified exactly, particularly it is strongly influenced by environmental effects. Therefore you can set up a well-known and fixed reference distance within your system and derive the speed of ultrasound from it, like in the Cricket Compass system [2]. If no calibrated distance exists and especially if the localization system works within common air, you can measure temperature and relative humidity just as well. Indeed, according to Bohn [16], the change in speed of ultrasound by relative humidity is just little in contrast to that by temperature. Sensing the latter property with adequate hardware is mandatory.

C. Ramping

As described for example in Mágori [17], another physical phenomena comes across with ultrasonic transceivers in general. Even if for instance the capture/compare block of a microcontroller's timer generates a clean ultrasound square wave to stimulate an US transmitter (as suggested in III-A), the transmitter itself does not transform this signal lossless into an acoustic wave, but needs a short settling time called *ramping*. This problem gets even worse at the receiver which amplifies the incoming wave much more sluggish. Fig. 3 shows a chirp detected by a capture/compare unit at a signal frequency f_{US} and the slow but clearly observable rising of its amplitude. This effect is subject to attenuation, i.e. it also depends on the traveled distance, and sporadically results in mistimed chirp detection. Hence the recorded point in time of US reception could be shifted in worst case by one period $\frac{1}{f_{US}}$ in either direction - even within an immovable system. Yet this would produce a distance error of about $\pm 8.6\ \mathrm{mm}$ at an ultrasonic frequency $f_{US} = 40\ \mathrm{kHz}$. Ramping is also the reason, why it is necessary to generate the chirp for longer than the length of one ultrasonic period. Raju [18] for instance uses a 12-cycle burst for one chirp, and so does SNOW BAT.

Fig. 3 also displays the time lag Δt_{delay} between real signal reception by the capture/compare unit used and the recognition

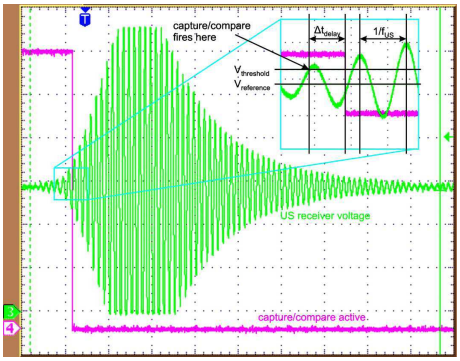

Fig. 3. Ramping at a receiver of 40 kHz ultrasonic chirps

of its interrupt at the time, when capture/compare gets inactive again. This delay is nearly constant but must be taken into account for the algorithm later on. Also noteworthy is the threshold voltage $V_{threshold}$ for the capture/compare unit and the reference voltage $V_{reference}$, around which the US receiver voltage oscillates. To avoid false detection of noise as a chirp, both voltages $V_{threshold}$ and $V_{reference}$ must not be too close to each other. For instance the MSP430 microcontroller [15], [19] is deployed at SNOW^5 sensor nodes [7] and allows a much finer adjustment of the reference voltage compared to the threshold. This manner is common practice with most other microcontrollers.

D. Multichirping

To increase accuracy, a mobile node can chirp more often for each measurement. That means, after the first chirp the mobile node sends a fixed number of further ultrasonic chirps at specific intervals, which are also defined within the CAV. The receiving anchors thus have to determine at least one such chirp and average over all received signals of that measurement. This result will be replied to the mobile one as the estimated distance. Thereby the average measurement error will be kept small, but the whole process will take longer and the mobile node only receives the average over all measurements. After this detailed analysis of TDoA based distance measurement and some involved physically given hardware effects, an algorithm for position estimation will be developed.

IV. From distances to positions

Several algorithms are available to derive positions from measured data. A couple of them solve a system of linear equations using matrices (and their transposes) like Reichenbach et al. [20] or Savvides et al. [5]. Some consist of trigonometric functions for adjustment of a coordinate system like Peng et al. [21] or for triangulation like Čapkun et al. [22]. By using such complex mathematical structures, you should keep in mind, that computational power as well as memory size of most microcontrollers used within WSNs nowadays is very restricted.

Thus, this section develops an algorithm which will handle the distance information gained in section III mostly without difficult mathematical expressions. Therefore, the system will be simplified first (IV-A). Some hints for low-level software design, e.g. approximation algorithm (IV-B), adequate data type selection (IV-C) and computational boosts (IV-D), (IV-E) will be given, too. To evaluate some of these hints, a short comparison between self-developed approximations and an already available algorithmic library is drawn (IV-F), before a proposal for position estimation (IV-G) closes this section.

A. System's simplification

By the past section, an anchor can determine its distance towards a mobile node and has to sent this distance information back to the chirping one. But for an accurate positioning some more distance information from different anchors is

necessary. The minimal number of distances required depends on the measurement method used as well as the dimension of the room to be observed and the desired accuracy. For example *trilateration* needs at least three distances to non-collinear anchors for two-dimensional positions and at least four distances to non-coplanar anchors for three-dimensional.

To simplify matters, all anchors s_i of the localization system are situated within a single ceiling plane C, which is parallel to the floor plane F, on which a mobile node m will move. The minimal distance between both planes is ceiling height h. With it, a static node s_i can not only determine the distance d_i between itself and the chirping one but it can also precalculate radius r_i of the projected circle on F (cf. Fig. 4), because specifying radii reduces the complex problem from intersecting (hemi)spheres to intersecting circles. On this circle the mobile node would only reside on under perfect conditions. According to the Pythagorean theorem, radius r_i results from the measured distance d_i and the known height h as follows:

$$r_i = \sqrt{d_i^2 - h^2} \tag{4}$$

B. Square root approximation

Adopting (4) as it stands would work just inefficiently, if the deployed microcontroller lacks mathematical hardware units to extract square roots. Linking an overcrowded math library like the *C math lib*, which emulates these difficult calculations by software, is sometimes not proper by reason of low memory (cf. section IV-F). However, for square root calculation some approximations exist, for instance *Newton's method*, an iterative alternative to approximately compute $\sqrt{a}$ for any $a > 0$ with an arbitrary initial value $S \neq 0$:

$$a_0 = \frac{S}{2},$$

$$a_{n+1} = \frac{1}{2} \cdot \left(a_n + \frac{S}{a_n} \right) \tag{5}$$

$$= \frac{1}{2} \cdot \left(\frac{a_n^2 + S}{a_n} \right) \tag{6}$$

This approximation converges with subject to S quite fast after just a few iterations. If the deployed microcontroller misses hardware floating-point devices – and a lot of them yet do so – integer mathematics has to be used anyway. For this reason, a difference less than or equal to 1 between new and old calculated result $a_{n+1} - a_n \leq 1$ serves as stop criterion. By using integers exclusively, some arithmetic operations will originate rounding errors, and the easiest way to reduce those is their prevention. To minimize rounding errors, one can switch to smaller units, i.e. multiply the corresponding values by a suitable constant as also suggested in Raju [18]. For example, instead of specifying every radius r_i in millimeter (mm), you can use hundredth millimeter or even micrometer ($\mu\mathrm{m}$) to soften rounding errors and to stay accurate to a millimeter at last. In doing so, you only have to keep these variables within their ranges and watch out for arithmetic overflows.

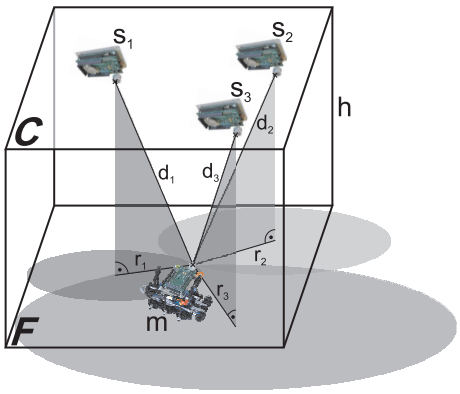

Fig. 4. Position estimation using distances d_1, d_2, d_3

In this account a software engineer has to trade off between (5) and (6). On the one hand, the first one could produce graver rounding errors compared to the second one, because of its earlier division $\frac{S}{a_n}$. On the other hand, the latter equation (6) may cause more overflows due to its square of a_n and subsequent addition. To gain less overflows and due to some microcontroller lack floating point numbers, (5) should be used slightly modified as

$$a_0 = \left\lfloor \frac{S}{2} \right\rfloor,$$

$$a_{n+1} = \left\lfloor \frac{1}{2} \cdot \left(a_n + \left\lfloor \frac{S}{a_n} \right\rfloor \right) \right\rfloor. \tag{7}$$

C. Data type selection

To save memory, these above-named (unit) expansions should be implemented moderate and well considered. For instance the Δt_{chirp} of a CAV (see section III-A) best symbolizes a period of time but not an absolute point in time. Using microseconds as smallest time unit, the data type `int` for this period just needs $2\ \mathrm{bytes}$ of memory but allows to describe up to sufficient $65.5\ \mathrm{ms}$. Whereas a point in time has to be at least of data type `long` or better of type `long long`, which would allow a validity period of more than 580.000 years, but requires $8\ \mathrm{bytes}$ of memory. Besides, for worse equipped microcontrollers big data types are broken down by compilers into types, natively supported by the hardware. As an 8-bit microcontroller can handle data types up to `char` directly, one `int` will be turned into two composed `char`.

D. Binary shifts

Equation (7) exemplifies another improvement as well, because in general binary right-shifts are executed faster than divisions, since a binary right-shift by one of an integer value is equivalent to a division by two, rounded off to the last integer. Usually, a multiplication by two is not significantly faster, e.g. the MSP430 emulates such a rotating arithmetic left-shift by an addition instruction (see the MSP430 family user's guide [15]). Besides, this microcontroller even offers a hardware multiplier, which works as peripheral device in parallel to its CPU and does not interfere with it.

For space optimization it is also advisable to pass most parameters of a subroutine as pointers, especially if the data types of the parameters allocate lots of memory like a `long long` does. For example a pointer at the MSP430 requires only $2\,\text{bytes}$ whereas a `long long` requires $8\,\text{bytes}$ of space. Thus, a space optimized implementation of a square root function written in C and using binary shifts as well as parameter passing by pointers can be found in listing 1.

```
unsigned long long sqroot(unsigned long long* a) {
 unsigned long long last = *a; // save passed parameter
 unsigned long long y = last >> 1; // y = ⌊last/2⌋
 while (last - y >= 1) {           // stop criterion
  last = y;                        // save old value
  y = (y + (*a) / y) >> 1;         // equation 7
 }
 return y;
}
```

Listing 1. Implemented square root function using simple initial value

E. Initial value selection

But this self-defined function `sqroot` requires much more time to extract the square root of an integer than the function `sqrtf` of the C math library (cf. table I). One main reason will be the badly chosen initial value `y`, which is the half of the given parameter `a` (cf. listing 1). Thus, each iteration gains just one single bit of information in worst case, e.g. a number of type `long` needs about 32 iterations. To speed up another and more adequate initial value has to be calculated, however its computation must not be too complex. Also using only integer values exclusively, this may be obtained by the initial value

$$a_0 = \left\lfloor \frac{a}{2^{\lfloor \log_2(a)/2 \rfloor}} \right\rfloor . \tag{8}$$

For its computation, the logarithmic function in base two is needed. But the design of the binary numbers allows a fast and adequate approximation for that logarithm. E.g., listing 2 presents a space efficient implementation and recycles the primary square root function from listing 1 at the same time.

```
// Searches for most significant "1" in a and
// thus returns i = ⌊|a|/2⌋ = ⌊log2 a⌋
unsigned int log2pa(unsigned long long* a) {
  // c points at a
  unsigned char *c = (unsigned char*)a;
  int i = 8;          // number of char in long long
  while (i--) {
    if (c[i] != 0) { // wanted "1" is within i's char
      c = (unsigned char*)((unsigned int)c[i]);
      i *= 8;         // search within i's char
      // search exact i
      while ((unsigned char)c >>= 1) i++;
      return i;
    }
  }
  return 0;
}

unsigned long long sqrootLog(unsigned long long* a) {
 unsigned long long last = (*a) >> (log2pa(a)>>1);
 long long y = last >> 1;
 while (last - y >= 1) {
  last = y;
  y = (y + (*a) / y) >> 1;
 }
```

Listing 2. Implemented functions for logarithm and square root calculation

F. Optimum achieved?

This last approximation `sqrootLog` seems to be small and fast, so we compared it to its earlier version `sqroot` and the square root function `sqrtf` of the C math library. We compiled each implementation with equal compiler settings, whereon listing 3 required $3424\,\text{B}$, listing 4 required just $430\,\text{B}$, and at last listing 5 required $518\,\text{B}$ of memory. Because the `sqrtf` function is defined for `double` values only, no space optimization using pointers as parameters is possible and the result must be casted from `double` into `long long`, too. Furthermore, the software emulated floating point arithmetic is automatically included when using the C math library, therefore lots of space will be allocated. And of course, the additional logarithm function requires a bit more space than the other version of the self-defined square root function, using the simple initial value.

```
#include <math.h>
int main(void){
 long long x = 0x1234567890ABCDEF;
 x = (long long)sqrtf(x); }
```

Listing 3. Square root function of `math.h`

```
int main(void){
 unsigned long long x = 0x1234567890ABCDEF;
 x = sqroot(&x); }
```

Listing 4. Square root function using simple initial value

```
int main(void){
 unsigned long long x = 0x1234567890ABCDEF;
 x = sqrootLog(&x); }
```

Listing 5. Square root function using complex initial value

For time requirements, we measured the average time need for 100 consecutive calculations of several exemplary positive numbers of type `long long` (cf. table I). It seems as if the calculating speed of the library defined square root function is higher than the self-defined one with the simple initial value, except for the number zero. The other self-defined square root function uses a logarithmic initial value and thus requires a bit more time for computation on smaller numbers. But on huge numbers, the last implementation is 2.5 times faster than the function of the C math library and even about 23 times faster than the first attempt of the self-defined square root function. Thus, you really have to find an application and device dependent trade-off between the requirements of space and time once more.

G. Position estimation

Granted that enough anchors answered their positions together with the measured distances and radii respectively towards the chirping node, the next challenge to bear for the mobile one is the interpretation of these facts. From radii and positions of the replying anchors, circles can be derived, indicating the potential positions of the mobile one. The intersection point of all these circles denotes the position where the mobile node resided during chirping. That, at least, is the theory. But via inexact hardware, ramping, rounding errors and other real world conditions, never all circles will

TABLE I
TIME REQUIREMENTS OF THE SQUARE ROOT FUNCTIONS IN µs

hex number	sqrtf	sqroot	sqrootLog
0x0000000000000000	297	12	26
0x0000000000000004	379	366	736
0x0000000000000009	380	720	738
0x0000000000000010	376	1073	1093
0x0000000000020000	488	3885	1098
0x000000000004C900	362	3844	1456
0x00000000F0000010	566	6173	1106
0x0000FFFFFFFF00C	901	8372	364
0x0000FFFFFFFFFF9	903	8699	365

intersect in one single point. For instance Tseng et al. [10] suggest the *maximum likelihood estimation* as a possibility for position estimation under uncertain data, but the *least squares technique* or the usage of a *Kalman filter* is also feasible.

The SNOW BAT system for instance uses another simple heuristic (cf. Fig. 5). Here, the circles are intersected with each other, around each intersection point is set a small tolerance zone. The more other intersection points are within such a tolerance zone, the more trustworthy are the points inside of it. If all circles of one measurement are intersected, the intersection points within the most trustworthy zone define a polygon, whose center of gravity determines the estimated position of the mobile node. With it, the localization process also withstands one faulty anchor among many others, however the more circles to intersect, the more precise the estimated position. But even if a single anchor node should fail totally because of material defect, computer bug, hard failure or just low battery, the remaining localization system still works properly, provided enough faultless anchors are within range.

V. CONCLUSION AND FUTURE WORK

This paper described some difficulties which could arise while the development of a just theoretically stated idea of a WSN for localization. It first mentioned some characteristics of such systems and specified TDoA in detail. Next it gave some hints and possible solutions, both for distance measurement and position estimation. Some hardware and software specific aspects were explained, for example the restricted data type usage of some microcontrollers, the dealing with ramping or the usage of integer numbers to minimize measurement errors and additionally save memory. It was also demonstrated that you have to find a trade-off between space and time requirements.

Most of the mentioned optimizations have already been implemented and tested on SNOW^5 sensor nodes [23], [7] within the so-called SNOW BAT system [6]. For this reason, it achieves an accuracy up to ± 1 mm for single distances and up to ± 4 mm for three-dimensional positions. But first experiments showed that the applied communication protocol is somewhat the bottleneck of such a wireless localization system, since a huge amount of coexistent chirping mobile nodes still should be able to localize at frequent intervals. Therefore further experiments have to be made, especially

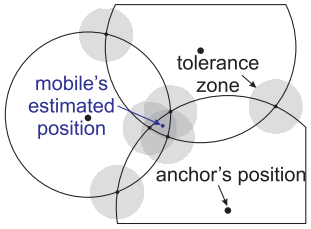

Fig. 5. Position estimation using three anchors

different communication protocols have to be tried out within this real world system. We also want to analyze the effect of an additional *field programmable gate array* (FPGA) to avoid some computational limits and easily extend the restrictive computation of the currently used sensor node SNOW^5. This would also open up new fields of activity like cryptographically secured sensor networks or video processing within a WSN based surveillance system.

REFERENCES

[1] A. Ward, A. Jones, and A. Hopper, "A New Location Technique for the Active Office," *IEEE Personal Comm.*, vol. 4, no. 5, pp. 42–47, Oct. 1997.

[2] N. B. Priyantha, A. K. Miu, H. Balakrishnan, and S. Teller, "The Cricket Compass for Context-Aware Mobile Applications," in *MobiCom '01: Proceedings of the 7th annual international conference on Mobile computing and networking*. New York, NY, USA: ACM Press, Jul. 2001, pp. 1–14.

[3] N. B. Priyantha, "The Cricket Indoor Location System," PhD Thesis, Massachusetts Institute of Technology, Jun. 2005.

[4] Y. Fukuju, M. Minami, K. Hirasawa, S. Yokoyama, M. Mizumachi, H. Morikawa, and T. Aoyama, "DOLPHIN: A practical approach for implementing a fully distributed indoor ultrasonic positioning system," in *UbiComp 2004: Ubiquitous Computing (LNCS 3205)*. Springer, 2004, pp. 347–365.

[5] A. Savvides, C.-C. Han, and M. B. Strivastava, "Dynamic fine-grained localization in ad-hoc networks of sensors," in *MobiCom '01: Proceedings of the 7th annual international conference on Mobile computing and networking*. New York, NY, USA: ACM Press, 2001, pp. 166–179.

[6] R. Kolla, M. Baunach, and C. Mühlberger, "SNoW Bat: A high precise WSN based location system," Institut für Informatik, Universität Würzburg, Tech. Rep. 424, May 2007.

[7] ——, "Snow^5: a modular platform for sophisticated real-time wireless sensor networking," Institut für Informatik, Universität Würzburg, Tech. Rep. 399, Jan. 2007.

[8] N. Bulusu, "Localization," in *Wireless Sensor Networks - A Systems Perspective*, N. Bulusu and S. Jha, Eds. Artech House, 2005, ch. 4, pp. 45–57.

[9] J. Beutel, "Location management in wireless sensor networks," in *Handbook of Sensor Networks: Compact Wireless and Wired Sensing Systems*, M. Ilyas and I. Mahgoub, Eds. CRC Press, 2005, ch. 20, pp. 1–23.

[10] Y.-C. Tseng, C.-F. Huang, and S.-P. Kuo, "Positioning and location tracking in wireless sensor networks," in *Handbook of Sensor Networks: Compact Wireless and Wired Sensing Systems*, M. Ilyas and I. Mahgoub, Eds. CRC Press, 2005, ch. 21, pp. 1–13.

[11] J. Hightower, R. Want, and G. Borriello, "SpotON: An indoor 3d location sensing technology based on RF signal strength," University of Washington, Department of Computer Science and Engineering, Seattle, WA, UW CSE 00-02-02, February 2000.

[12] P. Bahl and V. N. Padmanabhan, "RADAR: An in-building RF-based user location and tracking system," in *INFOCOM (2)*, 2000, pp. 775–784.

[13] N. B. Priyantha, A. Chakraborty, and H. Balakrishnan, "The Cricket Location-Support system," in *Proceedings of the 6th Annual ACM International Conference on Mobile Computing and Networking (MobiCom '00)*, Aug. 2000.

[14] J. Haartsen, "Bluetooth - the universal radio interface for ad hoc, wireless connectivity," *Ericsson Review*, no. 3, pp. 110–117, 1998.

[15] *MSP430x1xx Family User's Guide*, Texas Instruments Inc., Dallas (USA), 2006.

[16] D. A. Bohn, "Environmental effects on the speed of sound," *Journal of Audio Engineering Society*, vol. 36, no. 4, pp. 223–231, Apr. 1988.

[17] V. Mágori, "Ultraschallsensoren zur Abstandsmessung und Präsenzdetektion," in *Sensortechnik*, H.-R. Tränkler and E. Obermeier, Eds. Springer, 1998, ch. 10.2, pp. 511–553.

[18] M. Raju, "Ultrasonic distance measurement with the MSP430," Texas Instruments, Tech. Rep., Oct. 2001.

[19] *MSP430x161x Mixed Signal Microcontroller (Rev. D)*, Texas Instruments Inc., Dallas (USA), 2005.

[20] F. Reichenbach, A. Born, D. Timmermann, and R. Bill, "A distributed linear least squares method for precise localization with low complexity in wireless sensor networks," in *Proc. 2nd IEEE International Conference on Distributed Computing in Sensor Systems (DCOSS 2006)*, 2006, pp. 514–528.

[21] R. Peng and M. L. Sichitiu, "Angle of Arrival Localization for Wireless Sensor Networks," in *Proc. of the Third Annual IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks*. IEEE Computer Society, Sep. 2006, pp. 374–382.

[22] S. Čapkun, M. Hamdi, and J.-P. Hubaux, "GPS-free positioning in mobile ad-hoc networks," in *HICSS '01: Proceedings of the 34th Annual Hawaii International Conference on System Sciences (HICSS-34)-Volume 9*. IEEE Computer Society, 2001, p. 9008.

[23] R. Kolla, M. Baunach, and C. Mühlberger, "Snow^5: A versatile ultra low power modular node for wireless ad hoc sensor networking," in *5. GI/ITG KuVS Fachgespräch "Drahtlose Sensornetze"*, P. J. Marrón, Ed. Stuttgart: Institut für Parallele und Verteilte Systeme, Jul. 2006, pp. 55–59.

[24] M. Ilyas and I. Mahgoub, Eds., *Handbook of Sensor Networks: Compact Wireless and Wired Sensing Systems*. CRC Press, 2005.