Localizing mobile nodes in a relative coordinate system

Semester Thesis

Andreas Biri
abiri@student.ethz.ch

Information Security Group
Institute of Information Security
ETH Zurich

Supervisors:
Mridula Singh
Prof. Dr. Srdjan Capkun

June 21, 2017

Acknowledgment

I would like to express my gratitude to Mridula Singh for her supervision and guidance during this thesis. Her dedication and continuing support in the meetings were inspiring and resulted in countless interesting and constructive discussions. Without her invaluable ideas and insightful inputs, this project would not be where it is now. I hope that the present work will prove to be of great value for her future research. Furthermore, I would like to thank Prof. Dr. Srdjan Capkun for enabling me to conduct this thesis in his group and therefore allowing me to tackle such a fascinating topic.

Abstract

This thesis presents an infrastructure-less, scalable, real-time positioning system for mobile entities. The system is designed based on multidimensional scaling (MDS) and multilateration. One of the key features is that there is no need for fixed anchor nodes. We achieve this by leveraging multidimensional scaling to generate a relative 3D coordinate system, after which all nodes can be mobile. The system can support real-time position estimation of multiple mobile nodes with high accuracy. We evaluated systems using both simulations and a prototype implementation.The implementation achieves an accuracy of $\pm$30 cm and supports up to 40 moving nodes updating their position every second. This thesis is a first step towards evaluating the feasibility of building an infrastructure-less secure positioning system.

Contents

Acknowledgment **i**

Abstract **ii**

1 Introduction **1**

1.1 Motivation 1

1.2 Wireless sensor networks 2

1.3 Related works 3

1.4 Goals 4

2 System design **5**

2.1 Scientific model. 5

2.2 Design considerations 6

2.2.1 System architecture 6

2.2.2 MAC protocol 7

2.2.3 Algorithms 8

2.3 Means & methods 8

2.3.1 Hardware 8

2.3.2 Applied algorithms 9

2.4 System overview 11

2.5 Contention access period 13

2.6 Contention-free period 15

3 Implementation 18
3.1 Embedded System programming 18
3.1.1 Structure 18
3.1.2 Mathematical calculations 22
3.2 Visualisation 22
3.3 Verification 24

4 Evaluation 26
4.1 Error behaviour of the algorithms 26
4.2 Theoretical analysis 30
4.3 Measurements 33
4.4 General lessons learned 36

5 Conclusion & Outlook 38
5.1 Conclusion 38
5.2 Future work 38

Appendix 40

Bibliography 42

Chapter 1

Introduction

1.1 Motivation

"If you think that the internet has changed your life, think again. The IoT is about to change it all over again!" — Brendan O'Brien [1]

Over the last decade, an ever-growing number of smart, interconnected devices has emerged and has been put to use in fields ranging from scientific research through industrial systems to home entertainment. Recent technological breakthroughts in embedded computing, wireless communication and sensor technology allowed for the creation of large sensor networks with very low cost and power consumption. In clusters containing only a few up to networks interconnecting thousands of nodes [2], devices are placed in monitoring areas and cooperate in conducting measurements and aggregating information for processing. In order to do so, it is often essential to associate a position to the data for comparison and integration into an existing data set. Only with this knowledge, the event can be localized, mapped and tracked over time.

In recent years, various new sensor network concepts such as the Internet of Things (IoT) awoke the interest of the general public. Companies like Google started building dedicated operating systems and giants like Amazon and Verizon are contending for startups in the field to position themselves for the expected coming surge in applications and customers [3, 4]. With growing activity, developing appropriate hardware and specifically designed algorithms for such devices constantly gains in importance.

As the goal of sensor networks is to optain a considerable amount of different measurements and often require devices to be disposable, most such systems are inexpensive by design and are only connected over a single, relatively basic link. Ideally, one would like to use the same communication channel in another dual role for position estimation by doing distance measurements instead of having multiple separated wireless channels. Therefore, a system is required which can produce accurate localization of possibly mobile nodes without requiring complex and expensive hardware.

1.2 Wireless sensor networks

The term wireless sensor networks (WSN) is used to describe a set of wireless sensor nodes which collaborate to perform a given task by fusing individual measurements into higher-level information. Usually, WSN are self organizing networks consisting of heterogeneous, randomly placed devices. In most cases, the nodes consist of three components: a sensing subsystem used for data acquisition, a wireless communication subsystem for data transmission as well as an optional processing subsystem for local data processing [5, 6].

Wireless sensor networks are considered to be "one of the four large high-tech industries in the future" [7]. Their application spectrum covers area monitoring, environmental sensing (such as forest fireproofing, air pollution measurements and natural disaster prevention), healthcare monitoring (e.g. smart home), industrial process monitoring and control, military surveillance and traffic control [8, 9, 10].

From the single nodes, data is transferred to so-called *sink nodes* (base stations) through a multi-hop paradigm [8], where the data is aggregated for further processing (see figure 1.1). While an extensive amount of data is necessary for reliable data analysis, most applications rely on the measurement locations which have to be known *a priori* to set the gathered information into context. This information is then used to solve higher-layer problems such as location-aided routing and data fusion [9].

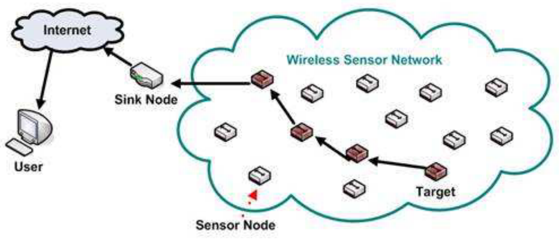


Figure 1.1: Wireless sensor network [11]

One way of providing this information would be to equip every single device with a GNSS (*Global Navigation Satellite System*) receiver. However, this solution is not feasible due to hardware costs which exceed the usual budget (with precise GNSS modules costing close to 100 dollars [12]) or size and power constraints [8]. Additionally, GPS reception is often strongly restricted or non-existent in urban areas, especially in indoor locations.

Another option is the precise disposition of mobile nodes by person. Because such an endeavour would require strict *a priori* knowledge of the deployment positions and is not scalable nor adaptive, this might only be applicable to very specific settings. Furthermore, it directly contradicts one of the core principles of a WSN by restricting movement and strongly limiting deployability and scale of the network.

For many applications, the absolute (global) position is not required. Therefore, it suffices to know the displacement of the measurement device in relation to other

devices inside the network. This is especially the case for clusters of networks where one is only interested in the interactions in-between nodes, such as drone swarms flying in formation and sensor nodes deployed over a coverage area by air for geological studies or when using location-aided routing [13]. This fact promotes the introduction of a relative coordinate system which does not rely on external infrastructure.
In case global coordinates are still necessary, they can easily be optained by equipping a dimishingly small fraction of the nodes with GNSS receivers. With only four such devices, the entire network with possibly thousands of members can be located and the global coordinates of each one of the nodes calculated [14]. In theory, this can even be accomplished in post-processing, e.g. after the nodes have been regathered and their last positions are known precisely. This method would even be realisable without implying any additional hardware costs at all.

1.3 Related works

Localization of WSNs has been studied extensively in the research community. The main effort herein has been focusing on infrastructure-based or -aided systems such as anchor-based algorithms, in which a (preferably) small percentage of the nodes (so-called *anchors*) are aware of their precise position in advance [14].
However, as mentioned above, for many applications, anchor-free algorithms are sufficient and require less dependencies and complexity. A majority of the publications for GPS-free systems has been concentrating on the 2D case [15, 16, 17]. However, an extension of those algorithms for 3-D positioning is difficult and suffers from large localization errors due to the complexity of environmental factors as well as node deployment in a further dimension [18, 24].
For the static case, one of the most promising approaches has been the introduction of Multidimensional Scaling (MDS) by Shang et al. [19] as MDS-MAP(C) to reduce localization error. This work has then been improved using patches of relative maps in the MDS-MAP(P) algorithm and an additional, optional refinement step [20]. In contrast to the former approach, this solution first calculates small local maps and thereafter aligns and combines them to a coordinate system spanning the entire network. Stojkoska et al. extended the principle to cluster-based MDS [21] and then implemented MDS-MAP for 3D WSN [22]. The concept of cluster-based MDS (CB-MDS) was analysed in 3D by Fan et al. [23]. Using improved heuristics instead of Dijkstra's algorithm [8] and a new refinement phase based on Levenberg-Marquardt (LM), a more robust solution called MHL-M was proposed by Saeed et al. [24].

Cui et al. [18] recently extended this concept for mobile nodes and applied an improved MDS algorithm for the tracking of nodes. Whereas localization only estimates the position of static nodes, tracking achieves the continuous localization of nodes over time [9]. For this purpose, the movement is estimated at each time step and historic location information is incorporated to increase the accuracy of the algorithm. Such an algorithm needs to be iterative to interconnect multiple independent measurements and create coherent position estimates. Examples of smoothing algorithms include (extended) Kalman filters and particle filtering; due to their strong reliance on a correct and *a priori* known system model such as noise distributions and node movement, these algorithms are not adaptive and only bear small resemblance to the real world [18]. Therefore, Cui et al. proposed an algorithm relying on polynomial data fitting for model-independent and adaptive accuracy improvement.

The algorithm proposed in [18] requires the computation of the entire distance matrix for each time step, i.e. measuring distances between all nodes for a single iteration. This method is not well scalable, especially for a small percentage of moving nodes, and takes too long for practical use in large scale WSN. Furthermore, it implies that during the measurements for a single step, all nodes are stationary, as otherwise data consistency is violated. To prevent those shortcomings, we propose a two stage algorithm: First, a relative coordinate system is created based on the MDS method, which can be improved by existing methods [18, 23, 24]. In a second stage, only the moving nodes perform ranging and update their position. This can be achieved using either trilateration or multilateration. Trilateration has already been deployed successfully for increasing the accuracy of the calculated positions of MDS [25]. The resulting positions are then enhanced using data fitting algorithms to provide smooth and accurate tracking results.

1.4 Goals

For this thesis, the goal was to develop a method which is able to calculate a 3D relative coordinate system for mobile nodes without relying on any anchor nodes or other infrastructure. The system should be able to give time guarantees to nodes in regard to position update frequency and allow multiple nodes to be mobile simultaneously. To our knowledge, there currently exists no other solution for infrastructure-less and precise localization and tracking using only Time-of-Arrival (ToA) measurements. In a second step, the algorithm was implemented on pre-existing hardware. The setup used an *Arduino Due* with a distance bounding IC by 3DBAccess [26].

The rest of this paper is structured as follows: *Chapter 2: System design* draws an overview over the design concepts behind the developed algorithm. This includes the designed MAC layer as well as the different stages of the protocol. *Chapter 3: Implementation* sketches the implemented solution to show the dependencies and the structure of the C code. Furthermore, additional supporting material written in Python and Matlab is presented and its usage explained. *Chapter 4: Evaluation* describes how the built system was tested under different cirumstances and quantified. In *Chapter 5: Conclusion*, we present our findings and show possible future extensions of this thesis.

Chapter 2

System design

In the following sections, an insight into the considerations behind the project as well as a detailed description of the two stages involved is shown. The underlying model and the criteria for devising and building the system are presented and their consequences analysed. Furthermore, we outline possible variations where one could extend the presented work by incorporating known algorithms to encrease accuracy and reduce data fluctuation.

2.1 Scientific model

Any scientific research relies on certain fundamental assumptions to abstract the real world and define a common basis to work upon. In order to maintain relevancy and broad applicability to the physical system, those conditions should remain limited. For this project, we built upon the following scenario:

- We consider a network of wireless devices moving independently in three-dimensional space without any supporting external infrastructure
- "One world": nodes have bidirectional links to all their neighbours which are not delimited by physical phenomenons such as shadowing or limited reach
- The nodes are homogeneous, i.e. they possess the same hardware and other technical characteristics
- Nodes can obtain information about whether they are stationary or moving (e.g. by using accelerometer data)
- The motion of the nodes is smooth (i.e. differentiable)
- Initially, some of the nodes are deployed simultaneously and remain static for the first few seconds. Additional nodes might join the network after this time has elapsed.

For the presented setting, the method should allow nodes to move freely and without central coordination, gather and maintain knowledge about its current position and that of surrounding neighbours and evade possible collisions at all times.

2.2 Design considerations

2.2.1 System architecture

As we are investigating infrastructure-free situations, the computation and localization has to be executed entirely on the nodes themselves. There are multiple ways to distribute the load between devices; considering the expected performance mentioned above, we chose a centralized approach with a master-slave relationship. This structure possesses the following advantages over a de-centralized approach:

- Guaranteed position update intervals enable us to enforce physical collision prevention for a given maximal velocity while still allowing dynamical adaptation of the number of measurements of an individual node (less moving nodes permit more rangings per node and an improved accuracy).
- Centralized channel control for certain phases renders selective network association possible (which offers the ability to prevent too many nodes from moving and joining the network).
- The master-slave relationship remains the same for the initial phase (in which MDS is applied) and the later iterative phase (while the nodes are moving).
- We gain the possibility to easily integrate a new node into the system. The integration is centrally coordinated by a master, who informs newcomers about the network state and containing nodes.
- Centralized updates allow detection of vanished or emigrated nodes (master node can periodically challenge all nodes to reaffirm their positions).
- The centralized control reduces the energy consumptions of the (numerous) stationary nodes, as they are only required to be active during the Contention Free Period (CFP) and can remain in standby for the rest of the cycle.
- All known (scalable 3D) algorithms for relative localization rely on beacons or other centralized algorithms such as MDS for execution; this choice therefore allows us to implement already known algorithms and test their performance.
- Coordination between multiple piconets/clusters is available, which makes large-scale communication in the network possible and could be used for improved scalability.

By design, all nodes share the same information. Therefore, even if for various reasons, a master should not be available anymore, another slave inside the cluster can immediatelly take over its position and seemlessly continue operations.

As we will see later on, one possible extension is applying CB-MDS, which takes advantage of clusters to distribute the load over multiple cluster-heads. For a scalable solution, this is unavoidable and the extension from this thesis straightforward. In this case, we would classify the architecture as *locally centralized*, as nodes only communicate in a restricted neighbourhood [14].

2.2.2 MAC protocol

One central aspect in any communication scheme is the selection of a fitting *media access control* (MAC) protocol. While multiple nodes compete for the same channel (as there only exists one in this scenario), data transmission is only successful if at most one node has exclusive access to the channel at any given time. The fair and efficient allocation of the channel is a challenging task, especially when the schedule should be dynamically created and adapted.

As the update frequency of the positions is critical for moving nodes and needs to be guaranteed to be above a certain frequency, a protocol which ensures that moving nodes receive regular slots of a certain length is unavoidable and therefore requires *time division multiple access* (TDMA). This project features a dynamic TDMA (D-TDMA) scheme with a *Contention Access Period* (CAP) (see Section 2.5) which allows mobile nodes to compete for the channel and request a slot. After a contention phase, the centralized controller (master) announces a *contention-free period* (CFP) (see Section 2.6), in which nodes which requested a slot in the previous CAP are scheduled in a TDMA fashion.

A contention-based MAC period is required, as anything more deterministically (e.g. only a CFP with classical TDMA) would not have allowed the network (which in practice may consist of thousands of nodes) to scale well when implemented in WSNs. Therefore, polling or token-passing algorithms were excluded. Other protocols under consideration for the CAP were:

- CSMA/CA: Not applicable, as the package length is too short for its usage
- CSMA/CD: Not implementable, as the hardware does not allow simultaneous sensing while transmitting
- ALOHA: Not implementable, as collision detection not available and we are limited to a single channel (no separate acknowledgments possible)
- MACAW: Implementing *Request-to-Send* (RTS) and *Clear-to-Send* (CTS) would not improve the system's performance, as the packages sizes used are in the order of the RTS/CTS packets themselves

As classical carrier-sensing is not possible with the present hardware, we entirely rely on feedback from the receiver (in this case the master node) for information about contention and occured collisions. This knowledge is transferred back to the slaves by an ACK package directly after reception. If no ACK is received, the node assumes a collision event and initializes a classical back-off mechanism. This method also solves other well-known problems such as the "Hidden Node" problem.

In order to keep contention periods (which are expensive in terms of time ressources) to a minimum, the algorithm allows nodes which already successfully gained channel access at least once to keep reserving a slot in future rounds. This is accomplished by including additional information in the position update broadcast (see Section 2.5). Furthermore, by guaranteeing slots to moving nodes, update frequencies can always be ensured and do not depend on congestion.

2.2.3 Algorithms

An overview over different localization algorithms has already been given in Section 1.3. For this thesis, a combination of multidimensional scaling methods for the initialization stage and multilateration for the iterative stage has been chosen. Compared to other systems, the main advantage of MDS localization algorithms is their utilization of the entire distance information at a single point in time to form a coherent coordinate system. Therefore, they do not suffer from error propagation throughout the system. Furthermore, in contrast to similar algorithms, there is no reliance on a set of reference nodes, whose movement would require the recomputation of all positions.

For the tracking of mobile nodes and an interconnection of position estimations over time, a polynomial data fitting strategy [18] has been selected. Other frequently seen algorithms for 3D range-based tracking are Kalman filters and variants thereof as well as particle filtering. However, these algorithms typically assume the presence of a system model (e.g. near-constant velocity models) and specific noise distributions which are known *a priori* (e.g. Gaussian distribution) [18]. As the measurement noise usually increases with the relative distance between nodes, it is signal-dependent [27]. Therefore, the usual assumption that the noise are i.i.d additive Gaussian random variables with identical covariance does not hold anymore. The present solution is not restricted to such cases.

As an alternative for multilateration, a *Maximum Likelihood Estimation* (MLE) can be implemented. To solve this non-linear equation, algorithms such as Newton-Raphson iterative method are used [28]. This can be problematic if the method does not converge to a global minimum and does not offer a fixed or predictable time consumption necessary for TDMA.
It has been shown that trilateration can be used to improve initial coordinats calculated by MDS [25]. To our knowledge, this work is the first one which extends this concept to mobile nodes in a three-dimensional space and utilizes multilateration for increased accuracy in a large node set.

2.3 Means & methods

2.3.1 Hardware

As a hardware platform for this thesis, we used a chip designed by 3DBAccess [26]. In combination with an *Arduino Due* microprocessor board, the *3DB6830B* integrated circuit allows for centimeter-precision ranging over ultra-wideband (UWB). For this, it employs a two-way ranging method based on Time of Arrival (ToA) measurements to calculate distances in-between multiple boards.
The code for this systems has been written for the current *MIDAS V2.0* software release using the programming language C. All interactions with the Arduino OS are handled over the built-in functions of the MIDAS platform.

2.3.2 Applied algorithms

Multidimensional Scaling

MDS originates from psychometrics and economics and is used to reflect the similarity of multi-objects. By reducing the dimension, one can visualize the data in a low-dimensional space and identify the underlying structure in-between objects [7].

Multidimensional scaling can be separated into four steps:

1. Calculate (or estimate, e.g. using iMDS [8]) the Euclidean distances between any two nodes to create a matrix of squared distances for the entire network.

$$D = \begin{bmatrix} 0 & d_{12}^2 & \cdots & d_{1|N|}^2 \\ d_{21}^2 & 0 & \cdots & d_{2|N|}^2 \\ \vdots & \vdots & \ddots & \vdots \\ d_{|N|1}^2 & d_{|N|2}^2 & \cdots & 0 \end{bmatrix} \tag{2.1}$$

2. Find the eigendecomposition of the double centered distance matrix by using *singular value decomposition* (SVD) to aquire eigenvalues and eigenvectors of the system.

$$D_{dc} = \frac{1}{2}TDT, \qquad T = \mathbb{I}_{|N|x|N|} - \frac{1}{|N|}.1.1^T \tag{2.2}$$

$$D_{dc} = Q\Lambda Q^T = PP^T \tag{2.3}$$

3. Use the decomposition to calculate the new positions in the relative coordinate system (RCS) by utilizing the three largest eigenvalues and their corresponding eigenvectors.

$$P = Q\Lambda^{1/2} \tag{2.4}$$

4. In an optional step, refine the relative coordinates by means of particle swarm optimization [18], least-square minimization [20] or Levenberg-Marquardt (LM) [24]. This increases the accuracy of the RCS and prevents strong fluctuations.

Trilateration

In order to calculate the global position of an object, any algorithm needs to rely on already known locations. One of the most common methods to arrive at such an estimate (which is also used for GPS localization) is called trilateration, in which the simultaneous distance measurements to exactly three known nodes is utilized [29]. In the case of error-free ranging, one can express the problem as the process of finding the intersection point of three spheres:

$$\begin{aligned} (x-x_1)^2+(y-y_1)^2+(z-z_1)^2 &= l_1^2 \\ (x-x_2)^2+(y-y_2)^2+(z-z_2)^2 &= l_2^2 \\ (x-x_3)^2+(y-y_3)^2+(z-z_3)^2 &= l_3^2 \end{aligned} \tag{2.5}$$

After solving the three quadratic equations (2.5), whereby $p_i = (x_i, y_i, z_i), \quad i = 1, 2, 3$ are the position of the three known nodes with IDs i and distance l_i to the node in question, one can directly obtain the position p_4 without additional data.

It directly follows that trilateration can also be applied to relative coordinates. The advantage of this algorithm is its simplicity as well as its low overhead and data consumption. However, as a position update merely relies on three other nodes and measurements in real applications are strongly influenced by noise, error propagation throughout the system can quickly cause deteriorating results and render the estimates unreliable.

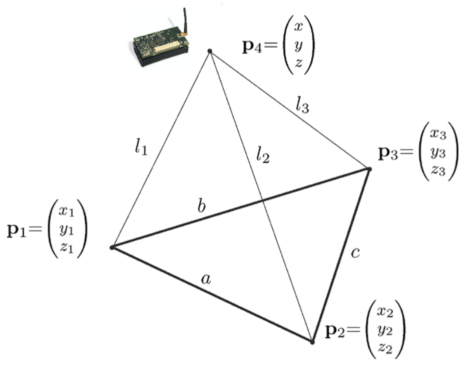

$\mathbf{p}_4 = \begin{pmatrix} x \\ y \\ z \end{pmatrix}$
l_1
l_2
l_3
$\mathbf{p}_3 = \begin{pmatrix} x_3 \\ y_3 \\ z_3 \end{pmatrix}$
$\mathbf{p}_1 = \begin{pmatrix} x_1 \\ y_1 \\ z_1 \end{pmatrix}$
b
c
a
$\mathbf{p}_2 = \begin{pmatrix} x_2 \\ y_2 \\ z_2 \end{pmatrix}$

Figure 2.1: Trilateration of node at position p_4 (adapted from [29])

Multilateration

One way of improving position accuracy and making estimates less error-prone is by increasing the number of known locations and distance measurements used in the calculations to decrease the influence of single erroneous data. By extending trilateration and exploiting an additional amount of existing information, the method should result in more reliable and precise position estimates. Multilateration implements this concept and is not limited to just three known sites like trilateration, but can incorporate as many positions as provided [30]. Therefore, the operator can directly influence the precision of the collected data according to his needs and adjust the balance between update frequency and localization robustness.

There are two major drawbacks in the presented solutions. Due to the nature of the calculations, a flip ambiguity exists which results in two different position estimates for each run of the algorithms. Whereas in many cases, the valid solution can easily be distinguished by a simple judgment criterion such as the requirement for positive

z-coordinates, this does not hold for the general case, especially for relative coordinate systems. However, by including an additional ranging measurement, it is still possible to uniquely distinguish between the candidates in most cases.
Additionally, both trilateration and multilateration rely on the single measurements to be taken virtually simultaneously (or rather, at the same location). For most physical systems, this ideal cannot be accomplished without disproportional hardware expenses. In this thesis, the staggered measurement points are of negligible influence, as their maximal spread in time stays below 10ms for five rangings, during which an object with a velocity of up to 100 km/h shifts by less than 25cm, which is below the effective measurement precision of the current hardware of 40cm.

2.4 System overview

We partition the process of building and maintaining a relative coordinate system (RCS) into two stages. In the first stage, following the model (see Section 2.1), nodes are assumed to be temporary stationary and allow for creating a complete description of the entire network at the current time instance. After wake-up and successful leader election (whereby one node is elected as master), the nodes start gathering information about their surrounding neighbours. This can be accomplished by implementing the following steps:

1. Perform ranging between neighbours (repeat multiple times and average results for improved accuracy and outlier reduction) and filter this using e.g. Kalman filters [31] to create a distance matrix for MDS
2. Send all distances to the master, who acts as cluster head
3. Estimate the non-neighbour distances using a heuristic approach (HA) in iMDS [8] or an iterative approach [32] instead of using Djikstra or Floyd to get a complete distance matrix (*optional: useful if "one world" assumption is dropped*)
4. Use MDS to get the relative coordinates by applying singular value decomposition (SVD) to the squared distance matrix
5. Improve MDS using particle swarm optimization (PSO) [18] or Levenberg-Marquardt (LM) [24](*optional*)
6. Merge multiple local coordinate systems from different clusters (if any) to one global coordinate system using D3D-MDS [23] (*optional: useful for improved scalability and if "one world" assumption is dropped*)

Steps 3 and 6 are not included in the current solution, as we fixed the scenario to a network where all nodes are in communication range with each other. Step 5 would increase the accuracy of the system, but is no integral part of the system and therefore not yet implemented due to time restrictions of the current thesis.

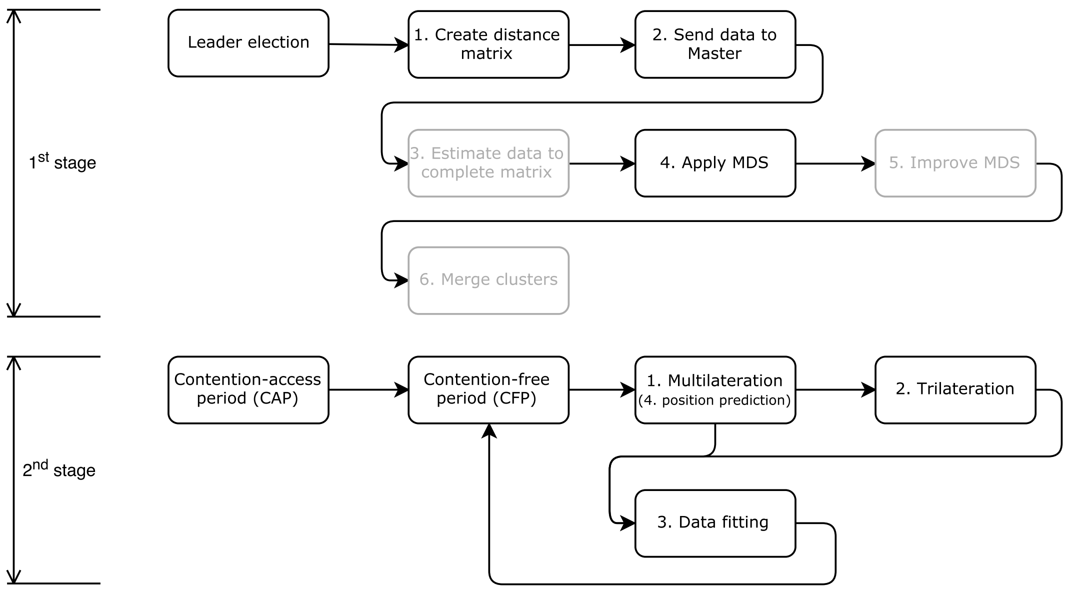

1st stage
Leader election
1. Create distance matrix
2. Send data to Master
3. Estimate data to complete matrix
4. Apply MDS
5. Improve MDS
6. Merge clusters
2nd stage
Contention-access period (CAP)
Contention-free period (CFP)
1. Multilateration (4. position prediction)
2. Trilateration
3. Data fitting

Figure 2.2: Visualisation of the two stages of the algorithm

In a second stage, the system transitions from stationary relations to a network containing mobile nodes. As MDS relies on simultaneous distance measurements, this algorithm is not well suited for uncoherent data and results in poor performance (see discussion in Section 1.3). This fact advocates the use of lateration algorithms which only rely on a subset of the distance information and still provide flexibility in terms of measurement accuracy. Motivated by this consideration, we implemented the following method:

1. Multilateration with n neighbours to update the node's position [30], whereby the parameter n can be determined dynamically by the node itself based on the number of other mobile nodes and the time allocated by the master for the CFP (see Section 2.6).

2. Due to numerical instabilities of the algorithm under certain conditions (such as when all points lie in the same plane), we used trilateration [29] as a back-up algorithm with reduced accuracy in cases where multilateration should offer no valid solution.

3. Apply data-fitting over historical data [18] to connect data points over multiple rounds. This smoothens the trajectory and increases correlation to the real positions by diminishing the influence of single rangings and interpolating over time.

4. Additionally, we exploit those parameters found in step 3 to predict the position over time. This enables us to accurately estimate the location of any node even in between own rangings and provides optimal data when providing distance measurements for other moving nodes. Therefore, we are free to im-

pose no restrictions on the amount of stationary nodes, as even moving nodes can offer precise location data for neighbours doing ranging.

As the nodes are only assumed to be stationary at the beginning, the first stage is just traversed once. The second one on the other hand is constantly cycled through; one completion is referred to as one *round*. At the beginning of each round, nodes that started moving will request a slot and gain access to the channel if the request is granted by the central authority, in this case the master (see Section 2.5). In a second *phase* after the contention-access period, the master will transmit a schedule in which the access sequence of the moving nodes that are granted access the channel is defined (see Section 2.6). One after the other, the moving nodes will now perform ranging with other nodes and update their positions. When all scheduled nodes are done, the round is over and a next one can be initiated by the master.

The entire system was specifically designed to have maximal modularity and allows one to adapt and extend the project gradually. This is crutial for having a flexible structure which can be optimized and adjusted to current hardware possibilities.

2.5 Contention access period

As described in Section 2.2.2, dynamic TDMA for the ranging uses a contention-free phase as well as a preceding period where moving nodes can contend for the channel. This allows dynamic slot allocations (compared to classical TDMA where a fixed schedule exist and is decided at the start), but still guarantees slots to the moving nodes which are already scheduled and therefore combines the advantages of both contention-based and contention-free protocols.

The beginning of each *contention access period* (CAP) is announced by the master who serves as a central scheduler. All nodes receive the same *Master Information Frame* (MIF) (see Appendix) which signals the start and contains information about its duration. The time allocated to the CAP is decided dynamically by the master based on the previous round: If many nodes tried to access the channel (and therefore, some might not have actually received a slot due to congestion), the probablility that a large number of nodes will retry this round is high and the CAP will be extended. However, if only few or even no mobile nodes contested the channel, we can limit this time to allow for more rangings during the following CFP (see next section). In order to ensure a certain CAP time and assure a minimum update frequency under large loads, the CAP has fixed minimal and maximal lengths which can not be exceeded.

After the nodes have received a MIF frame, they start an initial back-off to prevent immediate congestion at the start of the frame (see figure 2.3). Each node now decreases its backoff counter while the channel is idle. If the channel is already used by other nodes, this countdown will temporary pause (signaled by the dotted lign in figure 2.3); this prevents an increased probability of congestion after a transmission, as multiple nodes might have finished their backoff during the transmission of other nodes.

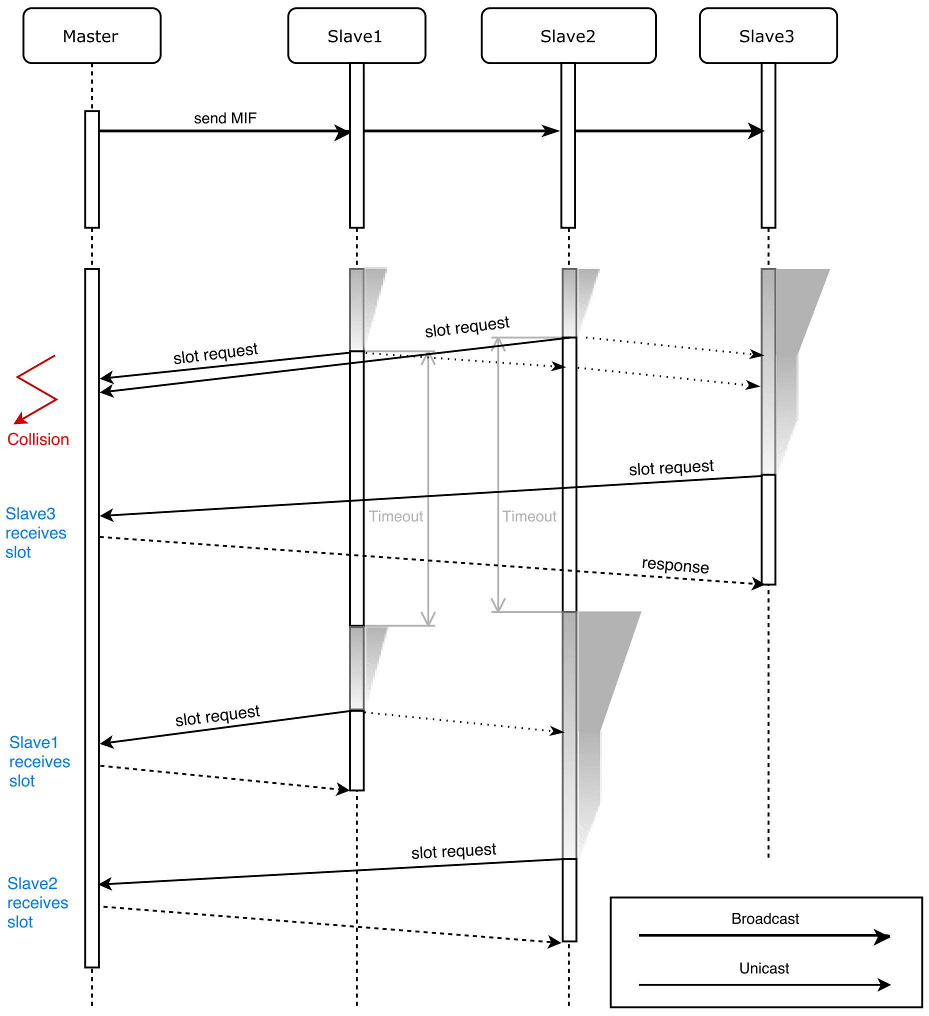


Figure 2.3: MAC protocol for the contention access phase (CAP)

When the counter reaches zero, the node checks whether the channel is free. This "collision avoidance" mechanism prevents sending while another node is waiting for a response from the master; however, it cannot detect immediate ongoing transmissions, as it needs to tap the channel for a certain duration.
If it is taken, it will wait for the transmission to end. Otherwise, it directly tries to access the channel. After the request is sent, the nodes waits for a fixed time (determined by the hardware characteristics of the node) until a timeout is reached. If it received an answer from the master, the master correctly scheduled the node in the next CFP and no further actions are required. In this case, the node has finished the CAP and is now waiting for the beginning of the CFP. The master on the other hand keeps listening on the channel for further requests until the CAP time has been reached.

If, however, the node does not receive an ACK, three things might have happened:

- Two nodes tried to access the channel at the same time
- A node tried to access the node at the same time as the master tried to send its ACK after receiving the packet correctly in the previous slot
- Due to other extern influences (moving objects, interference, etc.) the master did not receive the packet or the ACK got lost

In all cases, the resulting action will be the same: the node handles the case as a collision, as the channel is for one of the reasons given above not available at the moment, and initializes a back-off sequence. This can be seen in the second half of figure 2.3, in which two nodes independently start backing off after they previously collided. In order to prevent repeated collisions, the backoff timer is chosen at random inside an increasingly large backoff window to decrease the chance of further congestion.

As already mentioned in Section 2.2.2, nodes that gained a slot in a previous round and are still moving can signal this when sending their updated position in the "LOCATION" packet (see Appendix). The master will then automatically reschedule them without the necessity to use the CAP, which prevents unnecessary congestion and allows for guaranteed position update intervals. Therefore, already moving nodes have an assured slot as long as they require one.

2.6 Contention-free period

After nodes could apply for a slot during the CAP and had to compete with other neighbours, the next phase is strictly scheduled to have maximal efficiency. In a TDMA manner, each scheduled node receives a slot and can do ranging during its allocated period.

Similar to the CAP, a *contention-free period* (CFP) is initiated by a *Master Control Frame* (MCF), in which length of the entire CFP as well as the number of scheduled nodes and the entire scheduling is included. After receiving an MCF packet, each node knows when it is allowed to do ranging. From this point on until the end of the CFP, the master behaves just as any other ordinary node and has no special status; just as everyone else, it will be included in the schedule if it is going to do ranging in this round.

The procedure following a MCF is displayed in figure 2.4. The node which is scheduled first will create an additional schedule, in which it describes which nodes are going to be used for ranging to update its position. Each node will wait for the ranging schedule and decide whether its participation is required. If not, it will simply wait for the resulting update of the node's location. Otherwise, it will start responding to ranging requests of the scheduled node. Inside a response to a ranging request, it will send its updated position so that the moving node can use the most-current data for its calculations. If the answering node is also moving, this is

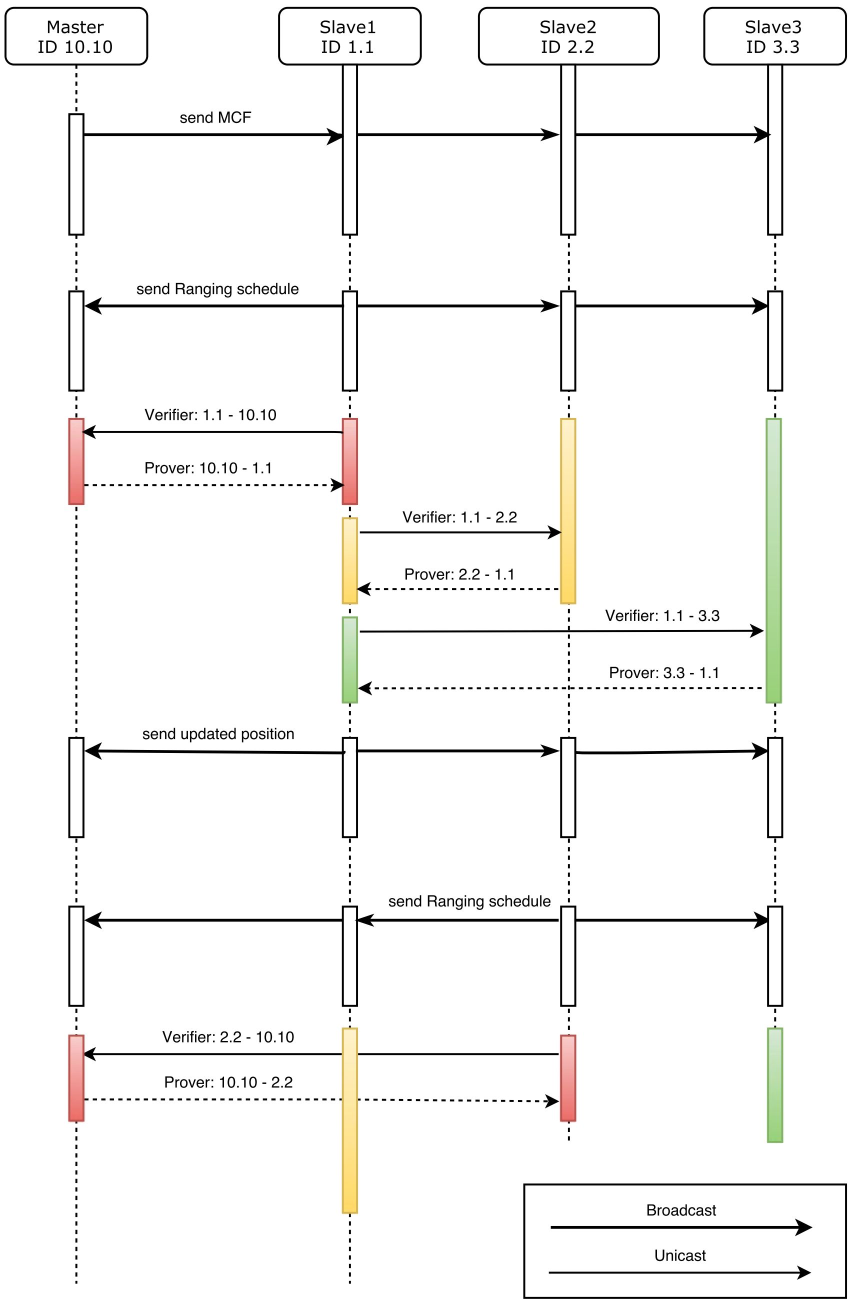


Figure 2.4: Contention-free period with ranging of multiple moving nodes

accomplished by using the data-fitting algorithm and predicting its effective position based on the results form its own last ranging.
After the ranging schedule has been completed, the node will recalculate its position. First, it will create a new estimate by executing the multilateration algorithm. Afterwards, it feeds this new data into the data-fitting algorithm and find a new position which fits into a smooth trajectory. As a last step, this new position will

then be sent to all other nodes to update their internal tables.

One by one, the nodes scheduled by the master will create their own ranging schedules, distribute them to their neighbours, perform ranging with the chosen nodes and then broadcast their updated position. The nodes can decide the total number of neighbours and the individual nodes they want to do ranging themselves. This should be based on the previous rounds and the position of the other nodes (optimally, the other nodes should be evenly distributed around the node to result in the highest accuracy). However, their time slot is limited and communicated to them in the MCF. Depending on the amount of moving nodes, the master can dynamically reduce or enlarge the total length. Just as for the length of the CAP, a maximal and minimal value has been fixed which the algorithm will not exceed.

Note: While a node which wants to initiate transmission needs to be in a so-called *Verifier* mode, reception can occur either in the *Prover* or the *Rx* mode. While in the latter, the node simply receives the packet, in the former mode, it automatically sends a reply back. This reply is used for the time-of-flight calculation which is then processed to calculate the distance between the nodes. In figure 2.4, the owner of the current slot is always in *Verifier* mode, while neighbours are in the *Prover* mode in case they are included in the ranging schedule and in the *Rx* mode otherwise, waiting for the position update.

Chapter 3

Implementation

In this chapter, we briefly cover the implemented algorithms as well as scripts written for visualisation of the data. In order to verify whether the actual algorithms on the nodes resulted in correct output, we cross-referenced them with output generated by a Matlab implementation.

3.1 Embedded System programming

The programming code for the nodes is written in C and uploaded to the *Arduino Due* as an Arduino project over the Serial port. The integrated curcuit used for distance measurement itself is neither programmable nor directly accessible. Data to and from the IC can be read using an API written by 3DBSystems. At the time of writing, the current software release for the hardware platform was *MIDAS V2.0*.

3.1.1 Structure

The code is partitioned into six main files, each with its own header. The dependencies between the files are sketched in figure 3.1 for the first stage and 3.2 for the second stage. The most important parameters for each file (situated in the corresponding header files) are included below.

structure.c is used as the entry point and, as its name suggests, defines the overall framework. In it, the parameters outlining the project as well as the fundamental structure for the different stages are given. It furthermore includes the data structure for information which is relevant to all other files such as node ID, current position and neighbouring nodes.
Its header *structure.h* is included in all other source files and grants access to global properties and functions.

- *PROJ_SUPPORTED_NODES*: limits the maximal number of nodes; this is only restricted for compile-time memory allocation.

- *PROJ_ID_RANGE*: defines the possible values for the service set identifiers (SSID); for static IDs, this should be adjusted.
- *PROJ_STARTUP_MOVING*: sets the node either as mobile or stationary; for this thesis, nodes cannot change their status during runtime.
- *PROJ_CAP_LENGTH_MAX*: together with *PROJ_CFP_LENGTH_MAX*, it defines the maximal update interval in-between position updates .

communication.c includes all functions used for inter-node communication. It also features MAC layer functionality and leader election. In most cases, there exists a sending and a directly corresponding receiving function.
Its header *communication.h* contains all constants which are immediately relevant to the transmitted data such as the definitions of the header types as well as timings required for correct interactions.

- *PROJ_SYNCHRONIZATION_DELAY*: sets a minimum delay at certain phases to synchronize the protocol over multiple nodes.
- *PROJ_COMMUNICATION_TIMEOUT*: defines a maximal waiting time for a response until the node automatically continues its execution and ignores the failure.

localization.c contains all functions related to multidimensional scaling, distance estimation and ranging. Internally, it utilizes **lateration.c** for the two implemented lateration algorithms (see section 3.1.2).

datafitting.c offers polynomial data-fitting and relies on Gauss-Jordan elimination (see section 3.1.2) to provide smoother trajectories. The implementation is designed to be modular and allows for quick inclusion and adaptation of other algorithms.

- *PROJ_DATAFITTING_ENABLE*: turns this option on or off.
- *PROJ_DATAFITTING_ORDER*: determines the order of the polynom used for interpolation; higher order requires drastically more computation time.
- *PROJ_DATAFITTING_HISTORY*: limits the number of historic values to be used.

helper.c provides helper functions for all other files and aggregates all debugging functionality. Furthermore, it is used for the visualisation.

- *PROJ_PRINT_ENABLE* allows switching output for the visualisation functions on or off to limit the load on the Serial port.

For figures 3.1 and 3.2, some dependencies were simplified or entirely omitted to improve recognition and readability of the general structure. The figures are directly comparable with figure 2.2 and should bear many similarities.

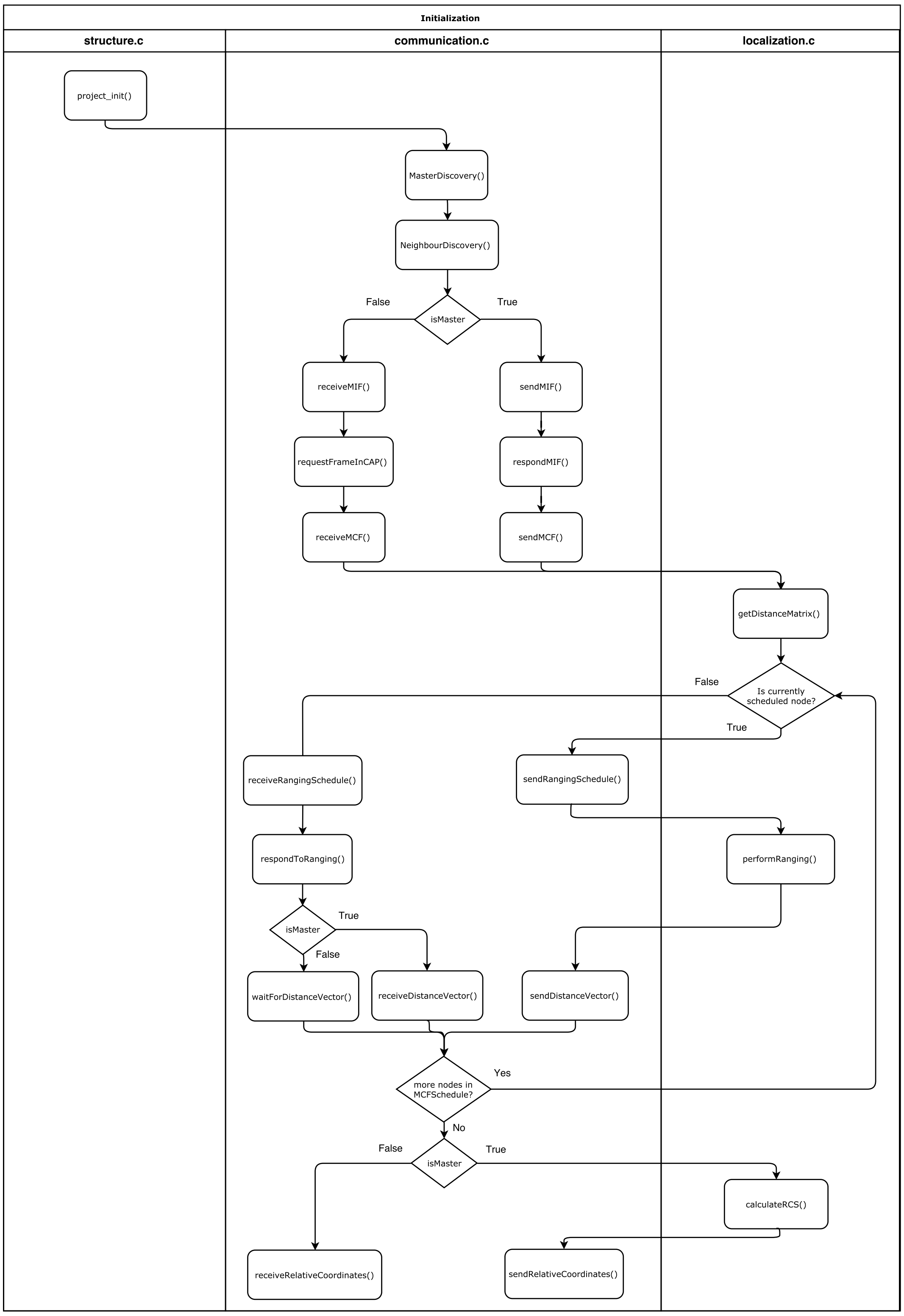


Figure 3.1: Function flow diagram for initialization

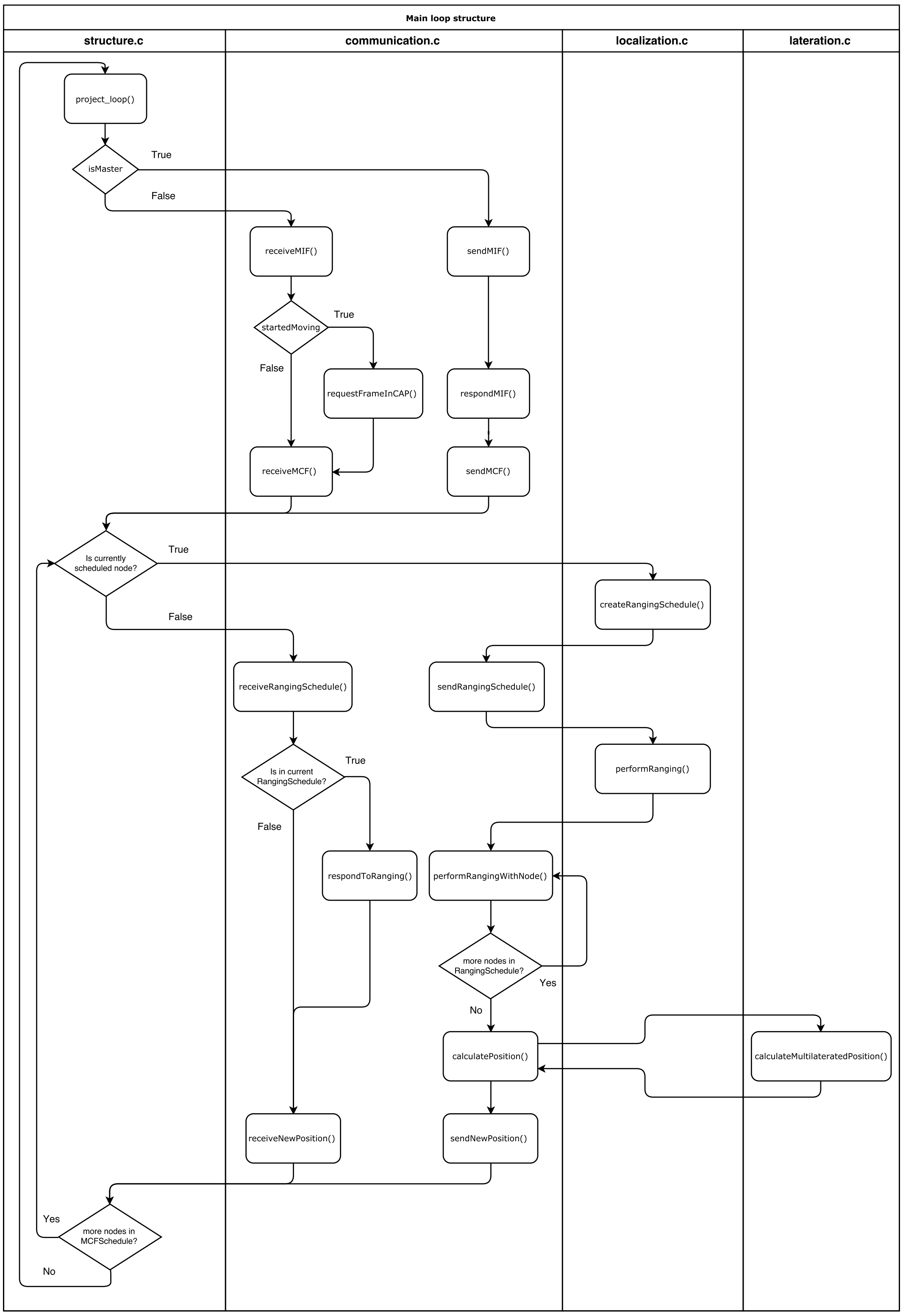


Figure 3.2: Function diagram for main loop structure after initialization

3.1.2 Mathematical calculations

Singular value decomposition

For multidimensional scaling, we require a singular value decomposition (SVD) algorithm. The code used for this thesis can be found in **SVD.c** and is an adapted version of code provided by Prof. Diana Crook [33] to handle *floats* and variable length arrays (VLAs).

Trilateration & multilateration

For trilateration, an algorithm developed by Thomas et al. [29] has been selected. As all terms used in the calculations have a geometric meaning, special cases such as singularities can be analysed in space, allowing to define direct implications.

Multilateration has been accomplished by utilizing work of Zhou [30]. In his paper, he gives a closed-form algorithm for a least-square approximation of the position which offers low computational complexity and allows a theoretical analysis of the performance under the influence of noise.

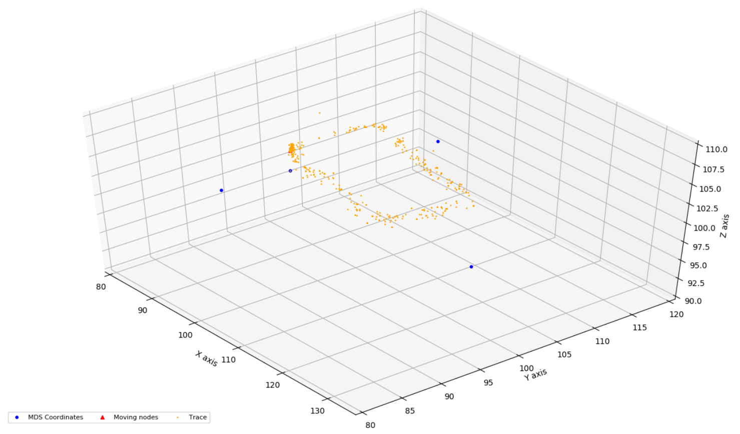
Gauss-Jordan Elimination

To solve the equation system for the polynomial datafitting algorithm, we apply classical Gauss-Jordan elimination and extend it with the Gaussian normal equations for over-determined matrices. As more historic values are required than the degree of the polynomial used, the equation system in the algorithm should always be over-determined. Otherwise, we would simply interpolate k points with a polynomial of degree $k-1$, resulting in the same data points as output as were used as input, rendering the algorithm useless.

3.2 Visualisation

To improve the analysis of the measurement data, we made use of the Python programming language to visualize the positions as point clouds in 3D with a library called *Matplotlib* [34]. This solution allows cross-platform data animation in real-time as well as logging the measurements for later usage and evaluation.

The implemented software has multiple features which can be enabled individually. During measurements, data is visualized directly in a 3D scatter plot and can be written into a log file on the device running the program. The measurement data can optionally be enriched by timing information to see duration and spacing of the measurements. During an experiment, it proved to be of great use to have the option of marking certain events such as reaching a specific point in the path or marking special data for later evaluation.



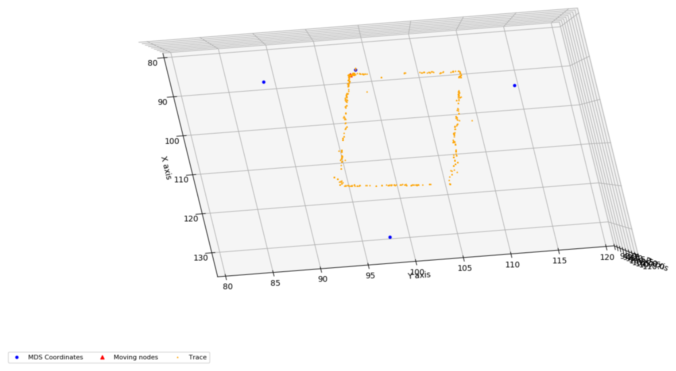


Figure 3.3: Measurements resulting from moving in a 15x30 meters square displayed in 3D (top) and from above for better recognition (bottom)

For post-processing and offline analysis, the option to read from a previous recording of an experiment has been included as well. Especially with tracing enabled, this allows to study the path and the measurement quality of the data even long after the experiment has been conducted.

Every node, independent whether master or slave, can be used in combination with the visualisation program. Transmission of the data from the node to the computer is accomplished over the Serial port. The reading is done in a non-blocking fashion to allow for smooth animation display and manipulation.
Data is encoded and includes information about whether they are MDS coordinates or ones from moving nodes as well as node IDs and 3D positioning data:

```
$  MDS: 02.02/128.047/098.002/099.094\n
$ DATA: 10.10/087.087/098.068/099.040\n
```

The program is started directly from console and only needs the current Serial port (in the example the Windows port *COM5*) as an argument:

```
$ python recvCoordinates_Serial.py COM5
```

Before execution, the above mentioned features can be set according to the current requirements:

- *PROJ_LOG_ENABLE* turns logging to a file on or off
- *PROJ_ADD_TIMESTAMP* includes a time stamp for each measurement if true; this can be utilized for time measurements
- *PROJ_READ_KEYPRESS* adds a separator into the log file to mark special events if true; this is used to create the ground truth for measurements
- *PROJ_READ_FILE* enables reading from a present log file instead of live data
- *PROJ_SHOW_TRACE* displace a trace of the moving nodes in addition to the current position if set to true

It is furthermore possible to save the animations directly as a video file. However, the animation is not affected when the coordinate system is manually rotated for a better viewing angle, wherefore we used a screen-grabbing tool to get the entire information and have more recording flexibility.

3.3 Verification

For correctness proofs of the implemented algorithms on the embedded system, we first built them using Matlab. This allowed us to properly debug the code on the nodes, as the platform does not provide direct debugging tools and requires output comparison. It also enabled us to quickly visualize and evaluate the applied methods with simulations.

mds.m creates a relative coordinate system based on multidimensional scaling. The realisation is based on code from [8] and [9]. A squared distance matrix D can be used as input to calculate the corresponding RCS for comparison with the output of the C code. Using **MDS_testing.m**, the algorithm creates random positions for a given number of nodes, calculates the RCS and compares the inter-node distances of the coordinate systems to verify the correctness of the solution.

tri.m is an implementation of the trilateration algorithm developed by Thomas et al. [29]. The code can be accessed on their webpage and is fairly compact. As input, it requires three points as well as the measured distances to the moving node. Due to the flip ambiguity discussed in previous sections, the algorithm outputs two possible solutions.

multi.m includes a realisation of the multilateration algorithm proposed by Zhou [30]. It requires numerous matrix calculations, but profits from smaller matrices than trilateration. Just as *tri.m*, the algorithm calculates the new position based on three or more existing node positions and the known distances to the node in question and outputs two possible locations.

Comparison.m is a script that can be used to compare results from trilateration and multilateration. It calculates both solutions and displays the resulting points. Furthermore, it shows the spheres around the previous positions to visualize their intersection and includes the base plane (through the three original positions) to show the flip ambiguity in 3D.
As can be expected, with just three nodes and perfect measurements, trilateration and multilateration result in the exact same coordinates. However, multilateration offers the ability to incorporate much more data and will result in better predictions for high-noise scenarios (see Section 4.1).

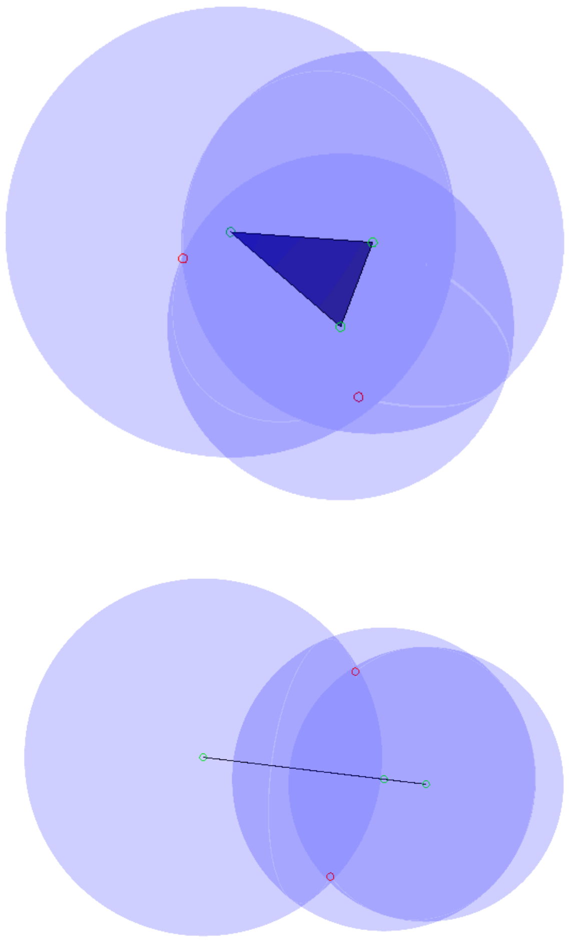


Figure 3.4: Display of the two possible solutions (red) of the multilateration algorithm; intersection of the circles (top) and the flip ambiguity along the base plane (bottom) are clearly visible

Chapter 4

Evaluation

This chapter shows the system in practise and gives an insight into how well it might perform under different circumstances. First, we simulated the algorithms with artificial noise, which allowed us to evaluate performance with various parameters and see the influences of different stages on the resulting measurement error. Then, we investigated the scalability of the system using the actual pysical parameters of our system. In the last section, the method was tested with four nodes and compared to a ground truth to evaluate performance in urban and suburban environments.

4.1 Error behaviour of the algorithms

In order to estimate the influence of distance measurement errors on the proposed system, we tested and simulated multidimensional scaling, trilateration and multilateration in Matlab under various conditions. We chose a network with 100 nodes randomly distributed inside a cube of side length 100m. We then introduced additive white Gaussian noise of zero mean and different variance and examined the influence on the accuracy of mentioned algorithms.
Note: All simulations were repeated a thousand times and the resulting histogram smoothened to increase readability. In order to facilitate comparison, the plots use the same x-scale; the probability density outside the given sector is negligible.

Multidimensional Scaling

MDS is an algorithm which is known to be prone to strong fluctuations if no post-processing of the relative coordinate system is conducted (as already mentioned in previous sections). Therefore, error examination is of great importance to see how well the system will perform.

In a first step, we considered the distance error in-between points of the RCS which will result from erroneous rangings. As can be seen in figure 4.1, the average distance error in the system increases linearly with the standard deviation of the noise and remains nearly constant over multiple iterations.

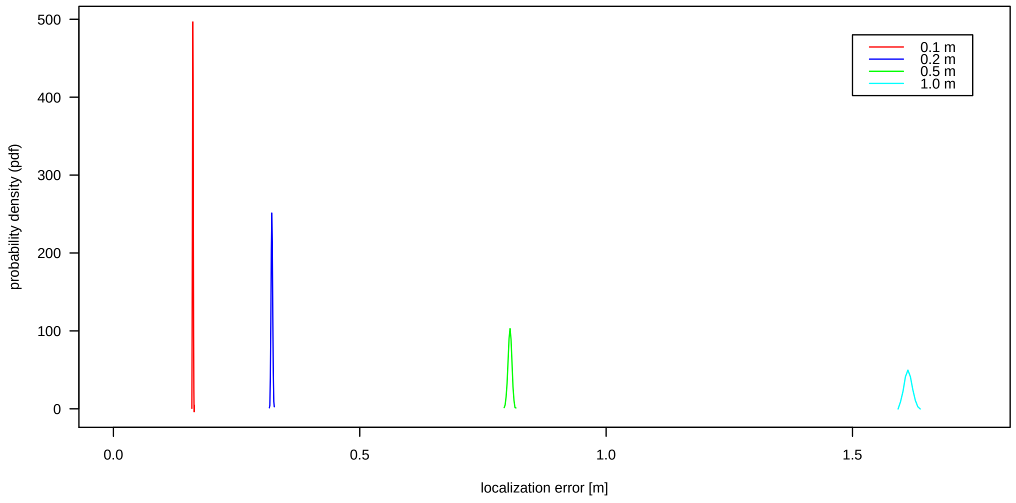


Figure 4.1: Distance error after MDS for measurement error with standard deviation 0.1m [red], 0.2m [blue], 0.5m [green] and 1.0m [cyan]

To gather information about the positioning error due to multidimensional scaling, we used the (defective) MDS coordinates in conjunction with the correct (noise-free) distances in the original coordinate system to calculate the position of a random point in the RCS (see figure 4.2). For small measurement inaccuracies, the MDS-induced error is fairly limited to a median value of 10-20cm. This increases up to 90cm in case of 1m standard deviation (see Appendix for more detailed results). Therefore, the influence of the MDS positions on the localization error is limited for the operational range of the system and still offers good performance even over large distances.

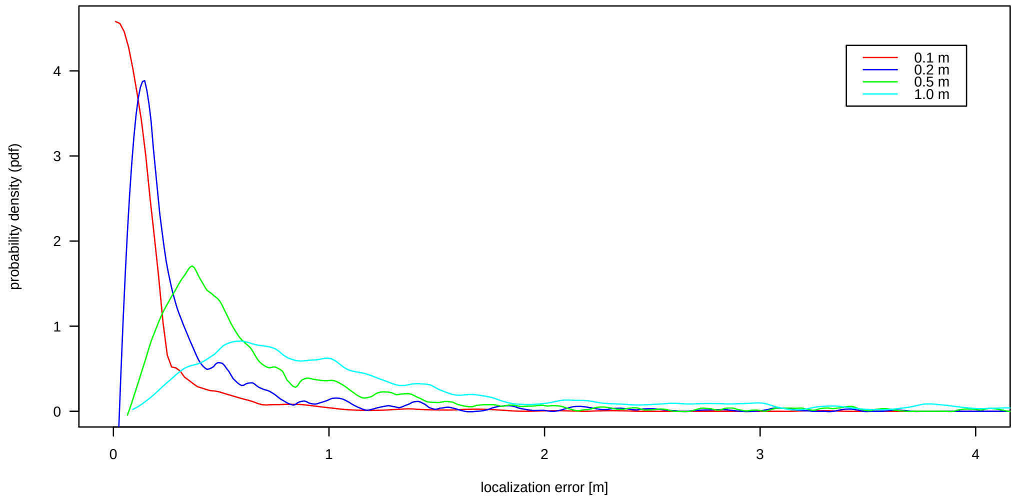


Figure 4.2: Distance between correct and calculated position with **MDS** for measurement error with standard deviation 0.1m [red], 0.2m [blue], 0.5m [green] and 1.0m [cyan]

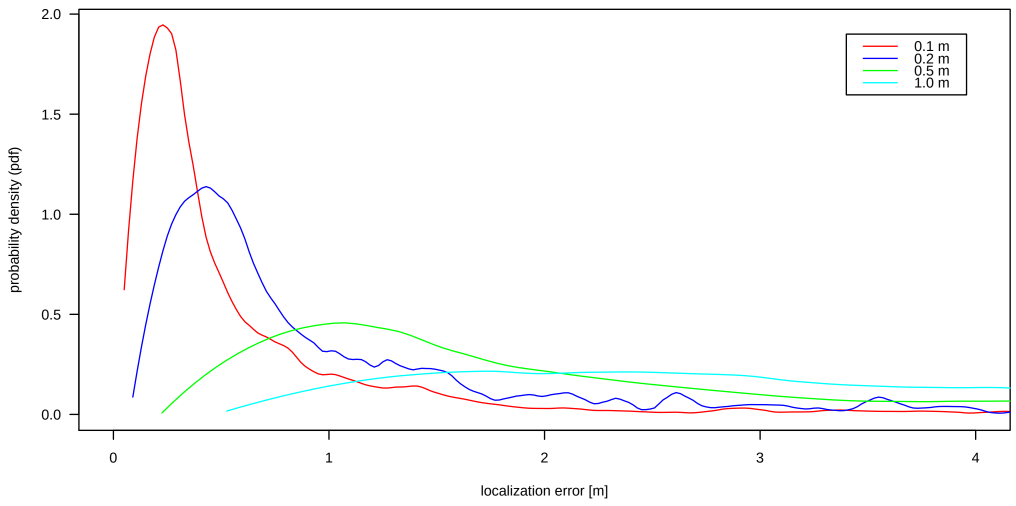


Figure 4.3: Distance between correct and calculated position with **Trilateration** for measurement error with standard deviation 0.1m [red], 0.2m [blue], 0.5m [green] and 1.0m [cyan]

Lateration

While trilateration offers the exact same results as multilateration for precise distance measurements, its lack of redundant data makes it prone to large localization errors. As this algorithm has just enough information to calculate the position, but no additional information to correct and interpolate data, ranging errors strongly influence the resulting solutions. Therefore, it is useful to compare trilateration and multilateration, which does not suffer from the same flaw and should therefore provide increased accuracy.

Figure 4.3 shows that position estimates quickly deteriorate and can only cope with a certain amount of measurement noise. While in the cases of only limited deviations, we can observe around half a meter of positioning error in most cases, the average error increases to nearly five meters with large noise.

Multilateration can handle this problem much better. Using ten nodes for ranging, figure 4.4 demonstrates that the localization error can be cut in half in comparison to trilateration. Even for a large noise with standard deviation of 1m, the median error still only lies at 1.6m and can be kept in check. However, the estimates still become notably more unstable and are spreaded over a much larger section when the noise level rises over a certain limit.

Another interesting analysis is the behaviour of multilateration under a varying number of neighbours used for ranging as seen in figure 4.5. Fixing the measurement error to a standard deviation of 0.5m, we increased the number of nodes included for the calculations and witnessed an expected increase in performance. Even though the noise level is on a rather high level, a large data set gives us enough redundancy to increase the robustness of the algorithm. Therefore, we can achieve a similar accuracy as the low-noise conditions by dynamically adjusting the ranging duration.

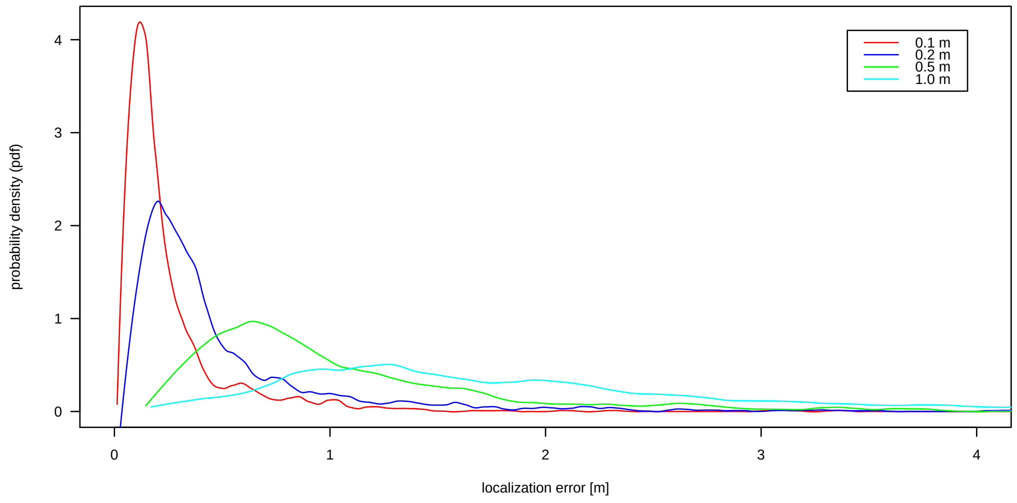


Figure 4.4: Distance between correct and calculated position with **Multilateration** for measurement error with standard deviation 0.1m [red], 0.2m [blue], 0.5m [green] and 1.0m [cyan]

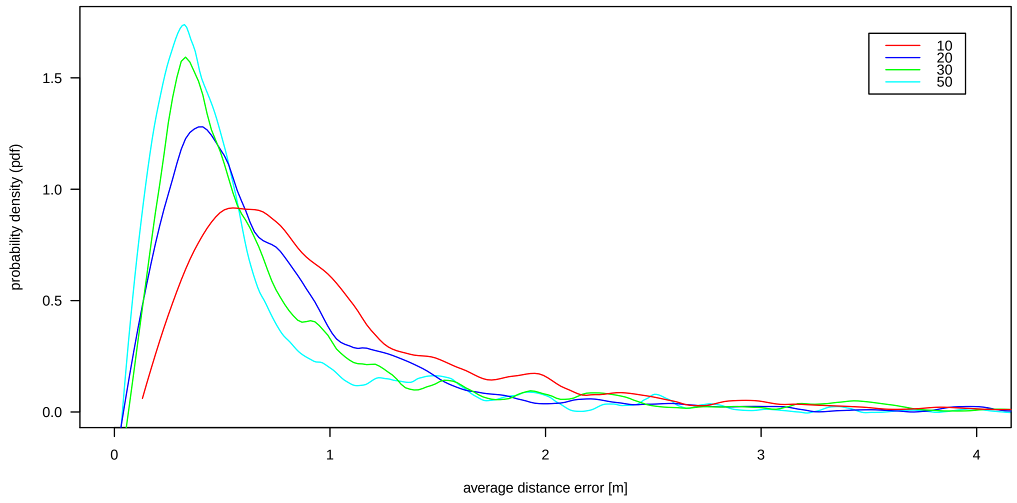


Figure 4.5: Distance between correct and calculated position with **Multilateration** when performing ranging with 10 [red], 20 [blue], 30 [green] or 50 [cyan] neighbours (measurement error fixed at a standard deviation of 0.5m)

4.2 Theoretical analysis

Of major interest for any party evaluating a proposed method is its scalability. Therefore, we examine the influence of the intended update frequencies and envisioned accuracy (depending on the number of nodes used for multilateration) on the amount of supported moving nodes. For this, derive a formula for the maximal number of moving nodes based on those two parameters. As the duration of the initial stage has no influence on the system's later performance and time consumption is assumed to be of no importance at the beginning, we concentrate on the later stage in this analysis.

Time consumption

First and foremost, some basic time durations need to be adressed. The presented values are all based on the currently implemented system:

- $\mathbf{t_{send}} = 2ms$: the time the system takes to send a value and ensure it is received correctly before it is allowed to send the next package
- $\mathbf{t_{switch}} = 3ms$: the time needed to switch in-between different modes of operation such as *Verifier* and *Prover* mode
- $\mathbf{t_{ranging}} = 2ms$: duration of one ranging attempt between two nodes
- $\mathbf{t_{calc}} = 1ms$: runtime of the multilateration algorithm
- $\mathbf{t_{fit}} = 5ms$: runtime of the polynomial data fitting algorithm (depends strongly on the chosen parameters)

It is interesting to see that multilateration, even though is uses more matrix multiplications, revealed to be faster than the trilateration algorithm. While multilateration with three nodes performed well with 320us for *float* values and 830us for *double* values, trilateration required 1520us due to the calculation of more determinants and larger matrices. Therefore, even though multilateration might seem to be more computationally expensive at first, it demonstrated outperforming trilateration not only in versatility but also computational complexity.

Theoretical bounds

Based on the timings in the previous subsection, we can define the duration of the entire system. However, we additionally require the expected number of participating nodes in the system. For this, we define three variables:

- $\mathbf{N_{CAP}} = 1$: the expected number of new nodes which need a CAP slot
- $\mathbf{N_{CFP}}$: the expected number of moving nodes using the CFP
- $\mathbf{N_{rang}} = 5$: the average amount of ranging partners of a moving node

With $N_{CAP} = 1$, we maximally have a small number of collisions in each CAP. Therefore, it suffices to expect at most one collision on average. As we start with a backoff-window of 8 and double it after a collision, we set a standard CAP length of $8 + 16 = 24$ slots with t_{send} duration each:

$$t_{contention} = (8 + 16) * t_{send} = 24 * 2ms = 48ms$$

Using those values, we can now define the durations of CAP and CFP as well as the total duration of one round:

$$\begin{aligned} t_{CAP} &= t_{send} + t_{switch} + (8 + 16) * t_{send} + t_{switch} \\ &= t_{send} + 2 * t_{switch} + t_{contention} \end{aligned}$$

$$t_{CFP} = t_{send} + N_{CFP} * \big(t_{switch} + t_{send} + N_{rang} * t_{ranging} + t_{calc} + t_{fit} + t_{send}\big)$$

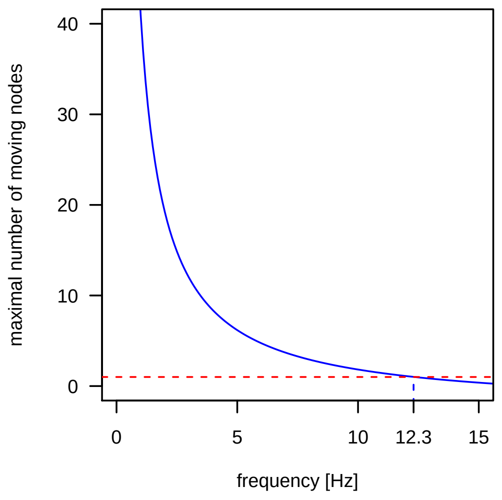
$$t_{round} = \frac{1}{f_{update}} = t_{CAP} + t_{CFP}$$

Using those terms and rearranging the terms, we can derive a formula for the maximally allowed N_{CFP} depending on f_{update} and N_{rang}:

$$N_{CFP} = \frac{\frac{1}{f_{update}} - t_{CAP} - t_{send}}{N_{rang} * t_{ranging} + t_{switch} + 2 * t_{send} + t_{calc} + t_{fit}} \tag{4.1}$$

We can see that both f_{update} and N_{rang} are inversely proportional to N_{CFP}. Therefore, if we increase demands on the update frequency or number of rangings for a new position estimation, we limit the supported number of moving nodes in the system. This can be caused by highter node velocities, requiring more time to evade collisions and therefore a higher update frequency, or increased accuracy needs which lift the number of nodes used for multilateration.

Evaluating equation 4.1 for the two parameters f_{update} and N_{rang}, we see that the update frequency strongly limits the maximal number of moving nodes (see figure 4.6), whereas an increased number of nodes for multilateration has only limited impact for values below ten. The maximal frequency which still supports moving nodes with N_{rang} fixed at 5 is 12.3 Hz, delivering position updates every 80ms. On the other hand, for a fixed t_{round} of 200ms, we can support up to 64 nodes used for multilateration with a moving node.
The formula can also be applied directly to physical systems. Using a maximal velocity of 20m/s and taking 4m as a safety distance, below which two nodes should evade each other, we require an update frequency of 5 Hz. Using equ. 4.1, we can directly jump to the conclusion that at most six moving nodes can be supported at any time. On the other hand, a single moving node can use up to 64 nodes for multilateration, allowing for extremely accurate position estimation.



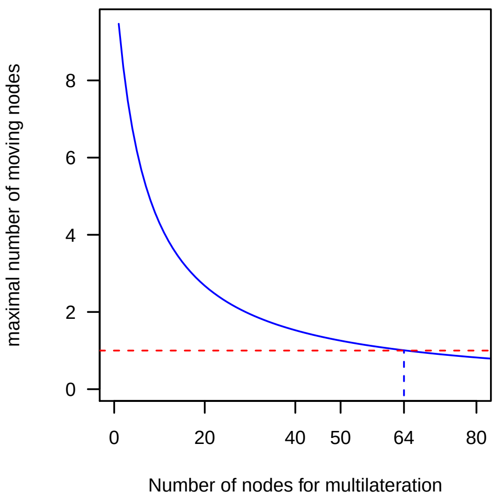


Figure 4.6: Evaluation of expression 4.1 over update frequency (left; N_{rang} fixed at 5) and over N_{rang} (right; f_{update} fixed at 5 Hz)

The time values chosen for this theoretical evaluation depend strongly on the physical implementation of the system. To investigate possible improvements by reducing certain durations, we considered the three parameters most likely to be enhanced by adjusting the hardware:

- decreasing the slot time used in a CAP from t_{send} to a value closer the ones used in comparable systems such as WLAN (802.11) with 9 microseconds by improving the recognition of partial frames.
- optimizing data fitting parameters to decrease t_{fit} to the same level as the time used for the position calculation itself instead of multiples of it
- enabling the same switching time between all modes by a software update to decrease t_{switch}. With the current hardware platform, switching between the receiving *Rx* and the sending mode takes more than one hundred times as long as switching from *Prover* mode to a sending mode, even though latter is technically more demanding and has additional functionality compared to firmer mode.

Following this reasoning, we decrease the corresponding values to $t_{slot} = 0.1ms$, $t_{fit} = 1ms$ and $t_{switch} = 0.1ms$. A comparison between figures 4.6 and 4.7 shows that the number of mobile nodes can be increased dramatically for a given update frequency by improving the hardware platform and optimizing the implementation to decrease the individual durations. However, the number of neighbours a moving node might use for multilateration does not have the same correlation and only slightly increases.
Such a system would allow fifteen nodes to simultaneously move and still update their position with a refresh rate of below 250ms. On the other hand, if requirements concentrate on accuracy, nearly 100 neighbours can be utilized to track a node with high precision.

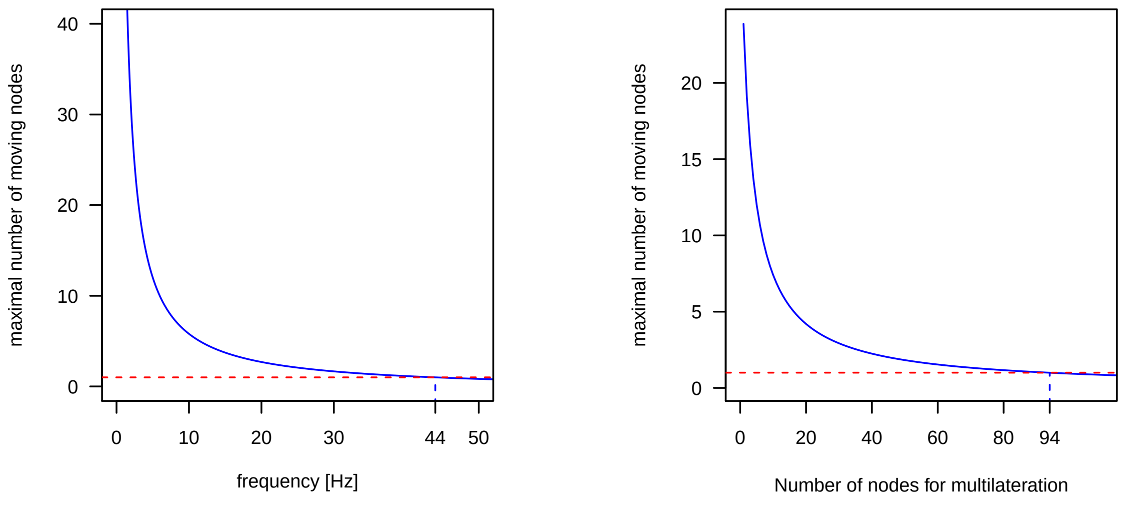


Figure 4.7: Evaluation of expression 4.1 over update frequency (left; N_{rang} fixed at 5) and over N_{rang} (right; f_{update} fixed at 5 Hz) with improved parameters

4.3 Measurements

To evaluate the system under real-life conditions, we took four nodes out into the field and let one moving node follow a preset path. This gave us a well-known ground truth as a reference value to compare the localization accuracy of the system with actual physical coordinates.
The nodes were setup in a height of one meter and had antennas facing into the center of the square. This prevented unreliable ranging results which are usually found when situating antennas near ground-level due to many direct reflections and when antennas have parallel normal vectors, preventing any line-of-sight components from reaching other nodes.

Urban

The first scenario covered a typical urban or indoors environment with many buildings and reflecting surfaces, leading to strong multipath behaviour. As all of those signals influence the measurements in different ways, the *line-of-sight* path (LOS) is more difficult to detect. This component is used to calculate the distance between nodes, as the estimation is based on the time of flight in-between them.

For the measurements, we used a 15m x 30m rectangle (see figure 4.8). The three stationary nodes were setup in a triangle around the rectangle at a slightly lower hight than the moving node to prevent all of them being in one plane. The distances between the nodes were 25m and 40m, allowing the moving node to be inside the triangle for most of the measurement series (which has an influence on measurement accuracy). For the positioning itself, no further setup was necessary, as everything is conducted directly by the system itself.

Error type	Average error	Median error
total	1.2396	0.8678
xy-plane	0.9689	0.6211

Table 4.1: Urban measurement results with outliers included

Error type	Average error	Median error
total	0.9346	0.8519
xy-plane	0.7265	0.6037

Table 4.2: Urban measurement results with outliers excluded

The resulting positioning estimates showed good results in this challenging environment. While the total average error was just above one meter, its median was well below at 86cm. Therefore, even with some outliers, the system achieves submeter accuracy when looking at the median distance error between estimated and real position.
As the movement was entirely restricted to the xy-plane, the z-direction only adds noise and therefore does not include any valuable information for the operator in this case. This motivates evaluating the measurement error solely projected onto the xy-plane. There, we see an average error of just one meter and a median value well below at only 60cm. Therefore, even over distances of 40 meters, our algorithm performed very well and delivered constant and accurate positions.

Due to a hardware bug concerning registers on the node, once in a while measurements may be rather off and outliers might occur. From over 400 measurements conducted for this test, excluding one percent of them as outliers with measurement errors of over twenty meters, the average error could be reduced dramatically by 25% to merely 90cm for the total error and 70cm in the xy-plane. Those values compare well with the median values and show that apart from one percent of the measurements, only a few of them are considerably higher than average and result in an average error being slightly highter than the median one.

Rural

To compare an urban environment with a rural setting, we reproduced the tests on a field. As the influence of multipath is strongly reduced, the second set of measurements can be used as a prediction for an upper-bound of the system accuracy in general.

Once again, the stationary nodes formed an isosceles triangle with 25m and 35m in-between. To prevent influences from objects around the nodes, we restricted the movement to a square of 10m x 10m. During the entire measurement series, the moving nodes stayed inside the triangle, which prevents strongly fluctuating results depending on the position.

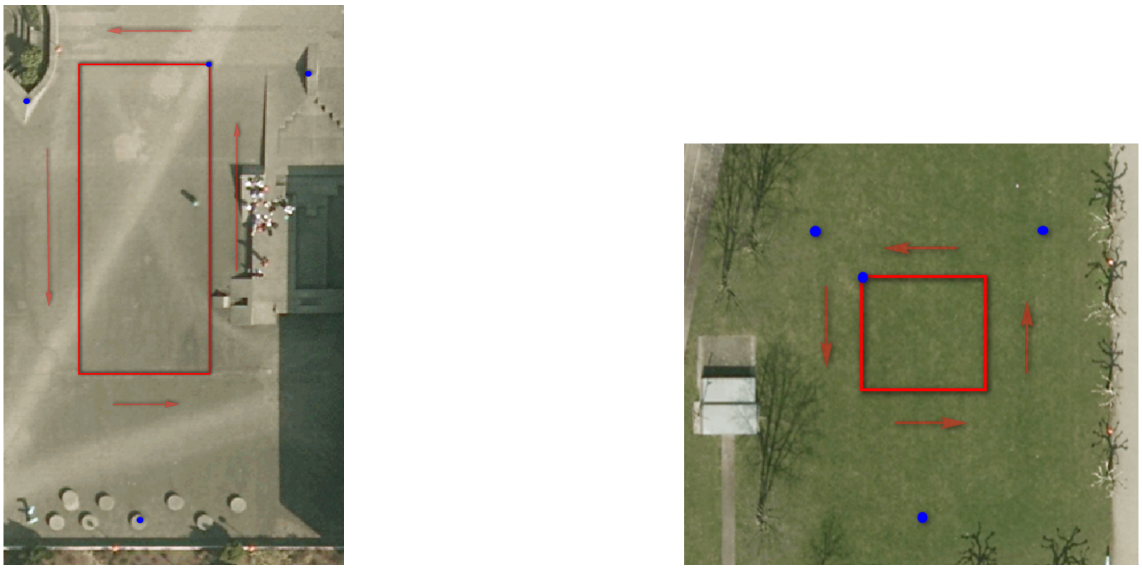


Figure 4.8: Urban (left) and rural (right) measurement setups; blue dots signalize stationary nodes and the starting position of the mobile node

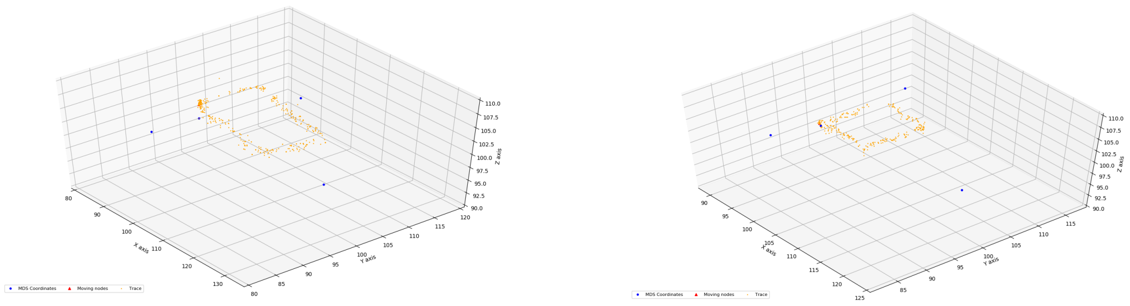


Figure 4.9: Urban (left) and rural (right) trace plots; blue dots visualize the MDS coordinates, orange triangle the past positions of the moving node and the red triangle the current (last) position

The new scenario proofed to result in better measurements, as was expected with less interference and therefore a better distance estimation. While the average error dropped by 40% to just above half a meter compared to the previous results without outliers, the errors in the xy-plane more than halfed to only 30cm. Furthermore, average and median errors are much closer together, suggesting even less outliers and more consistent measurements.

As can be seen in figure 4.9, a certain part of the measurement series had to be ignored due to the algorithm on one of the nodes being stuck. As in the case with the outliers in the urban environment, this is a result of a known current hardware bug which prevents nodes from receiving all the packages meant for them and therefore waiting an extended amount of time until a timeout is reached. During this period, the node cannot be used for ranging and therefore the moving node is unable to update its position with only two nodes left. This problem can be solved by either including more nodes in the test environment or improving the hardware directly.

Error type	Average error	Median error
total	0.5529	0.5293
xy-plane	0.3002	0.2628

Table 4.3: Rural measurement results

Multiple moving nodes

Due to issues with the present hardware, using a network with additional nodes to increase reliability and accuracy of the system was not possible, even though the system itself does not introduce any restrictions on this number. Unfortunatelly, a limited network consisting of only four nodes does not offer enough redundancy and robustness to have multiple moving nodes. As two moving nodes in this setting rely on each others position information for tracking, a slightly off estimate for one of them immediatelly results in the other node adjusting its position into the same directly and therefore strong error propagation. With more nodes, multilateration would prevent such directed behaviour by the "law of large numbers", as the errors would average out and a single measurement does not influence the rest as strongly.

4.4 General lessons learned

While testing, we encountered many surprising results and had to find means and methods to stabilize measurements and faciliating evaluation. Some of the most important practical experience gains are listed below.

- Even though measurements sometimes varied considerably from the real distances between nodes, multiple measurements proofed to be consistent most of the time. Therefore, repeating rangings multiple times is of less usage than using more nodes in multilateration. Best results are achieved when choosing the minimum distance in multiple repeated measurements due to the way leading-edge detection is currently implemented.
- Never measure directly at ground level. This distorts the measurements due to strong reflections and prevents reliable results.
- Ensure correct antenna alignment so that their normal vectors are never parallel to each other. Otherwise, their LOS components will not be detected correctly and the resulting measurements will not correspond to the real distances.
- Measurements in open areas without structures and reflecting surfaces delivered much preciser position estimates than urban or indoor areas. To ease evaluation and verification, use setups of appropriate scale, as otherwise measurement noise and displacement are difficult to keep apart.

- The flip ambiguity made measurements with multiple nodes extremely tiresome when only doing rangings with four nodes, as there is a 50/50 chance that the MDS direction and the chosen solution are aligned for each moving node. Therefore, using more nodes in additional planes is strongly encouraged to prevent this from happening.

- Multidimensional scaling can be quite instable and depends strongly on precise measurements, especially for networks with a small number of nodes. Therefore, we suggest using an improved MDS algorithm to eliminate those deficiencies.

- Multilateration is not necessarily computationally more expensive than trilateration as originally expected. Algorithms need to be implemented and tested on the present hardware to allow correct conclusions.

Chapter 5

Conclusion & Outlook

5.1 Conclusion

By relying on existing, proven algorithms and designing a custom system suited for our requirements, this thesis achieves the tracking of multiple moving nodes in a infrastructure-free environment. Through its modular design, it allows for selective and uncomplicated extensions and displays concrete improvement steps. We could show that in real-world applications, the system achieves satisfiable performances and can locate moving objects with high accuracy. With a refresh time of 200ms, up to six nodes can move simultaneously and can perform ranging with up to 64 other nodes to increase their precision. The nodes are deployed within minutes and can be monitored in real-time with intuitive and cross-platform tools to help identifying problems and estimate accuracy.

5.2 Future work

As to our knowledge, no other existing solution for infrastructure-less tracking with UWB measurements exists, this thesis served as a proof-of-concept and showed the method's capabilities and options. However, it still leaves room for improvement in multiple directions:

Optimization

- Evaluate the best suited parameters for datafitting. This should depend on the given movement patterns; as an example, erratic movement allows for less historic values to be used, whereas enduring trends can be better described with more historic values and lower polynomial order. This does however increase computational complexity dramatically and can have significant impact on the calculation time.

- Deploy refined hardware which resolves specific current problems. An example would be the increase of antennas to lessen the influence of the node orientation on the measured distances and to apply beamforming for increased range and signal-to-noise ratio. Furthermore, multiple internal problems can be addressed to prevent present workarounds and improve overall performance in terms of accuracy and time consumption.
- Apply MDS optimization methods (see section 2.3.2) to increase the initial accuracy of the relative coordinate system. Multidimensional scaling results tend to fluctuate strongly under noise, but can be improved in various ways as described in the thesis.

Scalability & Network dependence

- Find an optimal function to achieve the best ranging quality by dynamically deciding on the number of nodes to do ranging with. Currently, this is fixed to three nodes to get a maximum amount of rangings with single nodes due to a hardware register bug.
- Investigate which nodes offer the best accuracy for ranging measurements and how the number of nodes can be balanced with energy consumption. As all nodes which are not included in the ranging can sleep during this period, a trade-off between persistence of the system and accuracy has to be evaluated.
- Investigate the influence of different constellations on the precision of the position estimate and how nodes should be deployed or delocated to increase the performance of the system.
- Simulate large swarms of simultaneously moving nodes to evaluate the system's accuracy and stress test the implemented MAC protocols.

Extension

- Add functionality for a mobile node which newly arrives to be included in the system and aided during migration to other clusters. At the moment, this is not implemented solely because of the requirement for an additional phase and therefore further protocol complexity. The system itself is designed to be easily extendable by adding such a phase and already supplies most of the required functionality for such a step.
- Add recombination of multiple RCS to one large, global coordinate system by using rotation and alignment operations (see Fan et al. [23] for an existing 3D solution)
- Implement different systems such as anchor-based, GPS-based and infrastructure-free methods for the same measurements and compare their efficiency, required efforts, costs and accuracy.

Appendix

Header types

In *communication.h*, the different header types and their corresponding packet structure are described in detail. The table below lists the header types and their usage.

Binary	Abbrevation	Name	byte size
0000	SLOT_REQUEST	Slot Request broadcast (for CAP)	0
0001	DISTANCES	Slave Distance vector	X
0010	SCHEDULE	Ranging Schedule announcement	X
0011	LOCATION	Position & mobility status broadcast	7
1000	MASTER_ANN	Master Announcement broadcast	0
1001	MASTER_RCS	Master Relative Coordinate System	X
1010	MASTER_MIF	Master Information Frame	1
1011	MASTER_MCF	Master Control Frame	X
1100	MASTER_ACK	Master acknowledgment of a slave slot request	0

Table 5.1: Header types and their usage (X: size varies)

Deployment

For developing and testing the code on the nodes, we used an open source IDE called *PlatformIO* [35]. It offers user-friendly inclusion into Atom and easily allows node deployment and port detection via console.

Device listing and port detection:

```
$ pio device list
```

For deploying the program and reading the Serial output (e.g. on port *COM3*):

```
$ pio device monitor --port COM3 --baud 115200
```

Simulation results

The following table gives a detailed numerical insight into the plots mentioned in *Chapter 4: Evaluation.* Note that the average is strongly distorted due to outliers which can have errors of magnitudes higher than the median value.

Algorithm	Standard deviation	Average error	Median error
MDS	0.1m	0.2046	0.0971
	0.2m	0.3886	0.1927
	0.5m	0.8322	0.4761
	1.0m	1.5785	0.9383
Trilateration	0.1m	0.6449	0.3103
	0.2m	1.1181	0.6173
	0.5m	2.7455	1.5840
	1.0m	4.7429	2.1334
Multilateration	0.1m	0.2585	0.1581
	0.2m	0.5315	0.3250
	0.5m	1.1747	0.7911
	1.0m	2.1334	1.6203

Table 5.2: Statistical data corresponding to the plots in Section 4.1

For the comparison of multilateration with multiple nodes, we used a standard deviation of 0.5m. As compared to the upper results which used multilateration with a fixed number of nodes of ten, this table shows that increasing the number of nodes for the ranging results in a large accuracy gain.

Algorithm	number of nodes	Average error	Median error
Multilateration	10	1.0759	0.7692
	20	0.8215	0.6012
	30	0.7901	0.5095
	50	0.6289	0.4312

Table 5.3: Statistical data corresponding to the plots in figure 4.5

Bibliography

[1] "Knowledeblob: A brief about Internet of Things."
`http://knowledgeblob.com/technology/a-brief-about-internet-of-things-iot/`.
Online; accessed June 7, 2017.

[2] "IoT-LAB : Very large scale open WSN testbed."
`https://www.iot-lab.info/`.
Online; accessed June 7, 2017.

[3] "Techcrunch: Amazon has aquired 2lemetry to build out its IoT Strategy."
`https://techcrunch.com/2015/03/12/amazon-has-quietly-acquired-2lemetry-to-build-out-its-internet-of-things-strategy/`.
Online; accessed June 7, 2017.

[4] "Venturebeat: Verizon aquires IoT Startup Sensity Systems."
`https://venturebeat.com/2016/09/12/verizon-acquires-iot-startup-sensity-systems-to-make-cities-smarter-through-led-lights/`.
Online; accessed June 7, 2017.

[5] F. L. Lewis. *Wireless sensor networks.* Smart Environments: Technologies, Protocols, and Applications, chapter 4, 2004.

[6] K. Akkaya, and M. Younis. *A survey on routing protocols for wireless sensor networks.* Ad Hoc Networks, vol. 3, no. 3, pp. 325–349, 2005.

[7] C. Jun-jie, X. Ben-chong, and D. Li. *Relative Localization Systems and Algorithms for Wireless Sensor Networks.* IEEE International Conference on Networking, Sensing and Control, Sanya, pp. 1439-1444, 2008.

[8] Biljana Stojkoska. *Nodes localization in 3D wireless sensor networks based on multidimensional scaling algorithm.* International Scholarly Research Notices 2014, 2014.

[9] Sudhir Kumar, and Rajesh M. Hegde. *A Review of Localization and Tracking Algorithms in Wireless Sensor Networks.* arXiv preprint arXiv:1701.02080, 2017.

[10] X. Wang, M. Fu, and H. Zhang. *Target Tracking in Wireless Sensor Networks Based on the Combination of KF and MLE Using Distance Measurements.* IEEE Transactions on Mobile Computing, vol. 11, no. 4, pp. 567-576, 2012.

[11] `http://monet.postech.ac.kr/research.html`.
Online; accessed June 7, 2017.

[12] "ublox: Product display."
`https://www.u-blox.com/en/product-search/field_product_category/position-time-152/field_product_class/modules-199/field_product_tech/high-precision-gnss-171`.
Online; accessed June 7, 2017.

[13] I. Stojmenovic and X. Lin. *Loop-free Hybrid Single-path Flooding Routing Algorithms with Guaranteed Delivery for Wireless Networks.* IEEE Transactions on Parallel and Distributed Systems, 12(10), pp. 1023–1032, 2001.

[14] Adel M. Youssef, and Moustafa Youssef. *A taxonomy of localization schemes for Wireless Sensor Networks.* ICWN, pp. 444-450, 2007.

[15] S. Capkun, M. Hamdi, J.P. Hubaux. *GPS-free positioning in ad-hoc networks.* Proceedings of the 34th Hawaii International Conference on System Sciences, pp. 3481-3490, 2001.

[16] R. Iyengar, and S. Biplab. *Scalable and distributed GPS free positioning for sensor networks.* IEEE International Conference on Communications 2003, vol. 1, pp. 338-342, 2003.

[17] L. Wang, and Q. Xu. *GPS-free localization algorithm for wireless sensor networks.* Sensors, 10(6), pp. 5899-5926, 2010.

[18] W. Cui, C. Wu, W. Meng, B. Li, Y. Zhang, and L. Xie. *Dynamic Multidimensional Scaling Algorithm for 3-D Mobile Localization.* IEEE Transactions on Instrumentation and Measurement, 65(12), pp. 2853-2865, 2016.

[19] Y. Shang, W. Ruml, K. Zhang, and M. Fromherz. *Localization from mere connectivity.* ACM MobiHoc, pp. 201-212, 2003.

[20] Y. Shang, and W. Ruml. *Improved MDS-based localization.* INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies, vol. 4, pp. 2640-2651, 2004.

[21] B. Stojkoska, D. Davcev, and A. Kulakov. *Cluster-based MDS algorithm for nodes localization in wireless sensor networks with irregular topologies.* Proceedings of the 5th International Conference on Soft Computing As Transdisciplinary Science and Technology (CSTST '08), pp. 384–389, 2008.

[22] B. Stojkoska. *Nodes localization in 3D wireless sensor networks.* Proceedings of the 10th Conference for Informatics and Information Technology (CIIT '13), pp. 301–306, 2013.

[23] J. Fan, B. Zhang, and G. Dai. *D3D-MDS: a distributed 3D localization scheme for an irregular wireless sensor network using multidimensional scaling.* International Journal of Distributed Sensor Networks, 2015.

[24] N. Saeed, and B. Stojkoska. *Robust localization algorithm for large scale 3D wireless sensor networks*. Int. J. Ad Hoc and Ubiquitous Computing, vol. 23, pp. 82-91, 2016.

[25] S. Patil, and M. Zaveri. *MDS and trilateration based localization in Wireless Sensor Network*. Wireless Sensor Network, 3(06), p. 198-208, 2011.

[26] "3db Technologies: Proximity based Access control."
`http://www.3db-technologies.com/`.
Online; accessed June 7, 2017.

[27] A. Ribeiro, I.D. Schizas, S.I. Roumeliotis, and G. Giannakis. *Kalman filtering in wireless sensor networks*. IEEE Control Systems, 30(2), pp. 66-86, 2010.

[28] Y. Zhou. *An efficient least-squares trilateration algorithm for mobile robot localization*. Intelligent Robots and Systems 2009, pp. 3474-3479, 2009.

[29] F. Thomas, and L. Ros. *Revisiting trilateration for robot localization*. IEEE Transactions on robotics, 21(1), pp. 93-101, 2005.

[30] Y. Zhou. *A closed-form algorithm for the least-squares trilateration problem*. Robotica, vol. 29, p. 375-389, 2011.

[31] J. Wang, H. Li, X. Li, H. Ma, and Q. Huang. *The Accurate Estimations of Distances Among Nodes in Wireless Sensor Networks in a Complex Environment Based on an Adaptive Kalman Filter*. International Conference on Mechatronics and Automation 2007, pp. 735-739, 2007.

[32] V.K. Chaurasiya, N. Jain, and G.C. Nandi. *A novel distance estimation approach for 3D localization in wireless sensor network using multi dimensional scaling*. Information Fusion, pp. 5-18, 2014.

[33] "Iowa State University: paper by Prof. Diana Cook."
`http://www.public.iastate.edu/~dicook/JSS/paper/code/svd.c`.
Online; accessed March 28, 2017.

[34] "Matplotlib: Python plotting."
`http://matplotlib.org/`. Online; accessed June 12, 2017.

[35] "PlatformIO : open source ecosystem for IoT development."
`http://platformio.org/`. Online; accessed June 10, 2017.