Universidade de Aveiro Departamento de Electrónica, Telecomunicações e Informática, 2012

Malte Schmidt

Sistema de Localização para Hospitais com base em Tempo de Propagação

Hospital Localisation System using Round Trip Time Measurements

Universidade de Aveiro Departamento de Electrónica, Telecomunicações e Informática,
2012

Malte Schmidt

Sistema de Localização para Hospitais com base em Tempo de Propagação

Hospital Localisation System using Round Trip Time Measurements

Dissertação apresentada para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia de Computadores e Telemtica, Universidade de Aveiro, e Master of Science in Information Technology, Universidade de Ciências Aplicadas de Kiel, Alemanha, realizada sob a orientação científica do Prof. Doutor José Alberto Gouveia Fonseca, Professor Associado do Departamento de Electrónica, Telecomunicacões e Informática da Universidade de Aveiro e do Prof. Doutor Helmut Dispert, Professor da Faculdade de Informática e Engenharia Eléctrica da Universidade de Ciências Aplicadas de Kiel, Alemanha.

agradecimentos / acknowledgements

I would like to express my appreciation for the possibility to study and perform my thesis at the University of Aveiro and thank my supervisors Prof. Dr. Fonseca and Prof. Dr. Dispert for their continuous support. Furthermore, my thanks goes to my parents and family for their everlasting support, their advices and for giving me the possibility to realise my academic education.

palavras-chave

Localização clientes moveis, Tempo de retorno, Tempo de propagação do sinal, Trilateração

resumo

O presente trabalho discute a localização de pacientes em ambientes hospitalares e contém as seguintes partes principais: em primeiro lugar, é introduzido o estado de arte sobre técnicas de localização de modo a fornecer ao leitor os conhecimentos básicos para os capitulos seguintes. Em segundo lugar, é efectuada uma avaliação do estado de arte tendo em conta os requisitos necessários para ambientes hospitalares. Em terceiro lugar, é especificado um sistema de localização de pacientes com enfoque na sua arquitectura. A quarta e última parte contém uma proposta de implementação de um sistema de localização, fazendo referencia á investigação actual, sendo também descrito o software do servidor de localização que foi desenvolvido durante a tese.

keywords Mobile Client Localisation, RTT (Round Trip Time), Signal Propagation Time, Trilateration

abstract The present work discusses the localisation of patients in hospital environments and contains following main parts: Firstly, the state-of-the-art localisation techniques are introduced to provide the reader with the basic knowledge the following chapters are based on. Secondly, the state-of-the-art is evaluated regarding the requirements for a use in hospital environments. Thirdly, a blueprint design for a hospital patient localisation system, providing the key functionalities without looking at a specific implementation, is developed. The fourth and last part contains the proposal of a specific localisation system implementation by referencing to existing research and development as well as the introduction of a localisation server software that has been developed during the thesis.

Contents

Declaration

Unless otherwise indicated in the text or references, or acknowledged above, this thesis is entirely the product of my own scholarly work.

Date Signature

List of Figures

Chapter 1

Introduction

The introduction describes the context of the thesis, possible fields of localisation applications related to hospitalisation as well as its contribution to this fields.

1.1 Monitoring of high risk patients

As exemplarily depicted in 1.1 and 1.2 most industrial nations are facing an aging population and the consequent rise in healthcare expenditures. Furthermore, 1.3 illustrates the vanish of workforce resulting from the aging population and an entailing omission of people pay into health insurance. As seen at the diagrams especially Germany, Portugal and the Netherlands are affected by the need for young working people and rising costs in healthcare. This challenges for developing new ways of taking care for elderly people whether in hospitals or in their own homes. One approach for lower healthcare costs and increase the peoples independence is a continuous patient monitoring: Sensors in and on the patients body are measuring different vital functions and as soon as a threshold is reached an alarm is send to a central station, transmitting the actual recorded vital measurements and call for help. Furthermore, a continuous monitoring of special patients can help to early recognize possible problems. According to [25] "a large percentage of chronic diseases deteriorate to the point where a crisis is reached" and "that preventing occurrences of acute episodes holds the key to providing quality healthcare, reducing incidences of prolonged hospitalizations and resultant healthcare expenses". In general this and similar approaches can help to shift from a hospital centric system where every treatment is done in a central institution to a more patient centric system which allows the patients to stay at home more and more and where a domiciliary care by the relatives can be eased. Therefore, a technological supported healthcare could help both, the patients for having a higher quality of living and the insurances for saving money in a time of aging populations.

1.2 Localisation issues in patient monitoring

First and most obvious application for locating patients (whether in a hospital, a nursing home, at the home of the patients or everywhere else) is to be able to send help in emergency cases. The appeal for help could be send automatically as soon as a sensed vital functions exceeds a threshold or manual by the patient pushing an emergency button. Moreover it can be used for long time measurements to compare the location of the patients to its measured

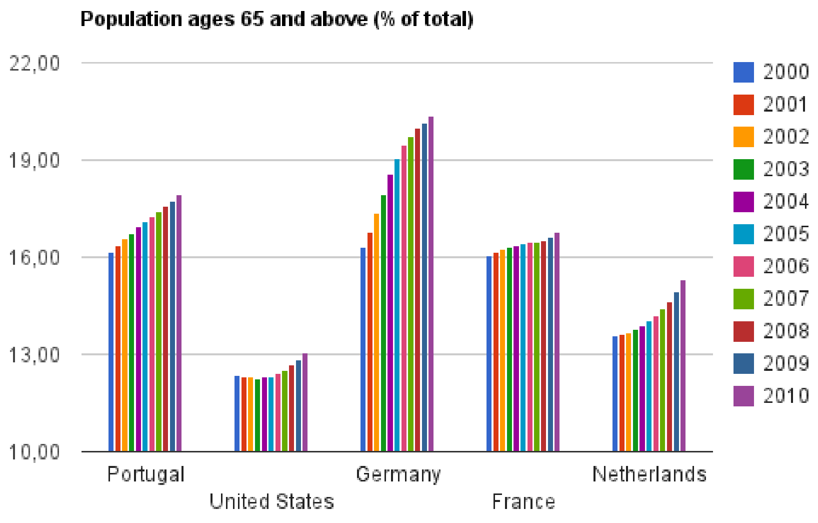


Figure 1.1: Aging population, Datasource: [2]

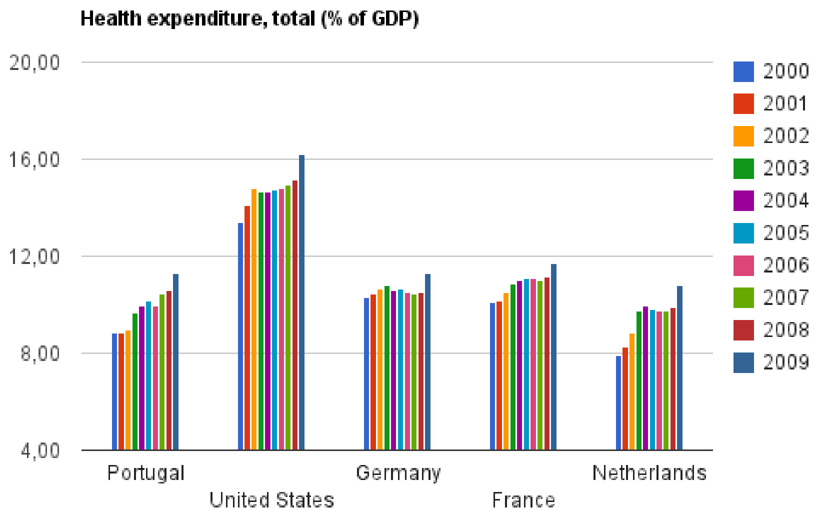


Figure 1.2: Rise in health expenditure, Datasource: [2]

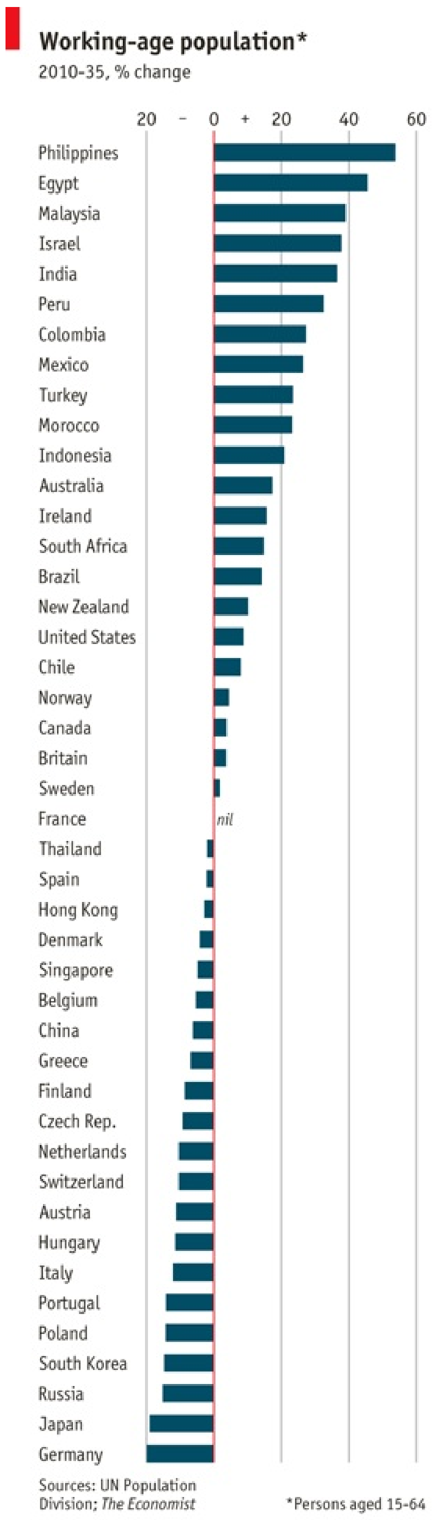


Figure 1.3: Change of working-age population, taken from [1]

vital functions for better understanding the patients issues. For example it could be interesting to know that every time the measured data shows that the patient feels better and is more relaxed, he sat in the nursing homes park.

But Independent of the application a comprehensive and reliable infrastructure which performs the localisation has to be provided. Since a breakdown can be life-threatening the system has to be failure-resistant. And if a defect occurs it has to be reported to the staff and to the users automatically to inform them that they can not build on it anymore and have to call help other way.

1.3 Contribution and content of the thesis

The thesis is summarising the state of the art localisation techniques and subsequently evaluates them regarding the application for the use case defined in chapter 2. Furthermore, the requirements for such a system are specified and a concept for a test system is drawn. The last part contains of building and evaluating a test system for the localisation of patients regarding the predefined requirements.

Chapter 2

Use case definition

The focus of this thesis is the localisation of people in hospitals or nursing homes. If patients have to stay in a hospital for medical observation they could be allowed to move free at the hospitals area to have a walk outside or visit the cantine. Similar situations occur in nursing homes where elderly people who aren't able to take care for themselves because of any diseases are nursed. In these cases, when the patients are not in their rooms and an alarm case is reported by a monitoring device it is important for the medical staff to localize the patient as fast as possible to be able to help. This alarm functionality could be implemented as an automatic mechanism initiated by a monitoring device that detects a complication or as a manual assistance call by pushing a button at a wireless device. The system which is to develop targets:

- A localisation in hospital / nursing home rooms or outside areas.
- An accuracy of a 2m radius by line of sight measurements.
- Cheap and flexible infrastructure to equip the rooms with.
- The use of common communication standards such as IEEE 802.11g.
- Use of state-of-the-art techniques and customization where needed.

Chapter 3

State of the Art and Related Work

The following chapter is the theoretical basis for the thesis and contains the state of the art in localisation approaches as well as related work in the field of localisation and some associated research topics. Firstly, two different localisation strategies are introduced followed by a description of the some widely used measuring principles. Afterwards the signal propagation delay measurement, which is essential when using certain measuring principles (e.g. RTT or TDoA) and the different influences on it are discussed.

3.1 Localisation Strategies

First, this section contains the definition of two localisation strategies which are to differentiate in this paper.

3.1.1 Infrastructure based localisation

The infrastructure based localisation depends only on the network and is independent of the client which has to be located. The advantage is that theoretically every client in the network can be spotted and no action of the user, like installing a software or setup the device, is needed. The obviously occurring problem is a possible breach of privacy if people devices are located without their knowledge or even movement profiles of individual users will be saved and evaluated. For example, a supermarket could be interested in how their customers walk through the market, where they linger or if they respond to advertised offers.

Despite the legal considerations an infrastructure based approach is very comfortable, even if an involvement of the user can't be bared at all. The action a user still has to perform is to accept a connection establishment with the wireless network.

3.1.2 Mobile Client involved localisation

The mobile client involved localization is based on an interaction of the network infrastructure with the user and his/her device. In respect of the use of a mobile phone this interaction could be the installation and use of an application, which interacts with the network and performs the positioning. An integration in any location based service is conceivable, too. This could be the provision of a map of a huge building complex like an airport or a university or a medical service, which is the focus of this thesis.

3.2 Measuring Principles

The following chapters contain the definition of and the introduction to some widely used localisation measuring principles. According to [11] the explained methods can be divided into two main groups: Lateration and angulation. Furthermore, the lateration can be divided into approaches using time measurements and the one measuring the signal strength at the localisation object (i.e. a Wireless LAN client like a laptop or mobile phone). See also figure 3.1.

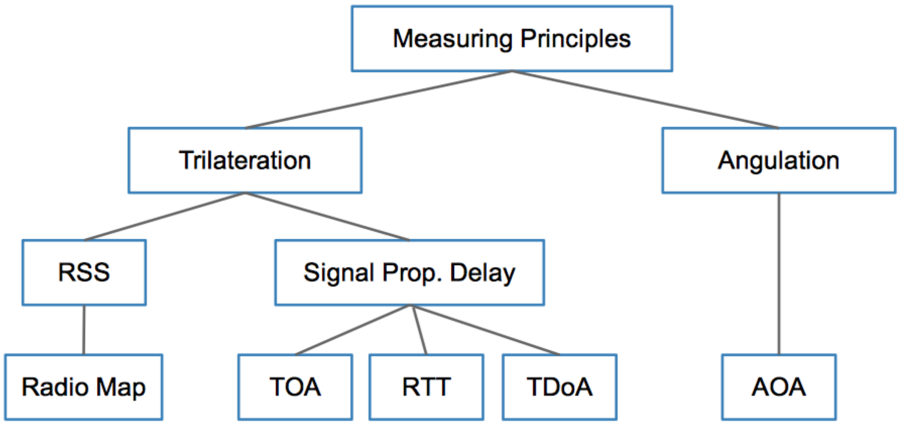


Figure 3.1: Overview of Measuring Principles

3.2.1 Trilateration Techniques

Trilateration is a localisation using the distance of the localisation object to three known reference positions (i.e. a wireless access point of the network infrastructure). The distance to the object can be described as a circle around the centre of each position. As depicted in figure 3.2 the object can be located at the intersection of the three circles. The approaches described in the following differ in how to measure those distances.

If the number of known points is two or especially if it is more than three the approach is also referred to as multilateration. Two reference positions result in two possible locations of the object from which one has to be barred. Additional references can be used to achieve a higher localisation accuracy, which has an extraordinary impact in non line of sight environments. But one has to avoid the case that all base stations are located in one straight line. In this case two opposite locations separated by the line on which the base stations are placed will be received. This avoids an uniquely localisation wherefore the base stations have to be spread into different walls of a room.

Measuring by using the Received Signal Strength

Using the received signal strength indicator (RSSI) is one possibility for measuring the distance between a client and a base station. The RSSI usually is a parameter of one byte (range from 0-255) which "is a measure by the PHY[1] of the energy observed at the antenna used to receive the current PPDU[2]" [28]. The RSSI is a relative parameter. "Absolute

[1]Physical layer (i.e. network interface card)

[2]PLCP (physical layer convergence procedure) protocol data unit

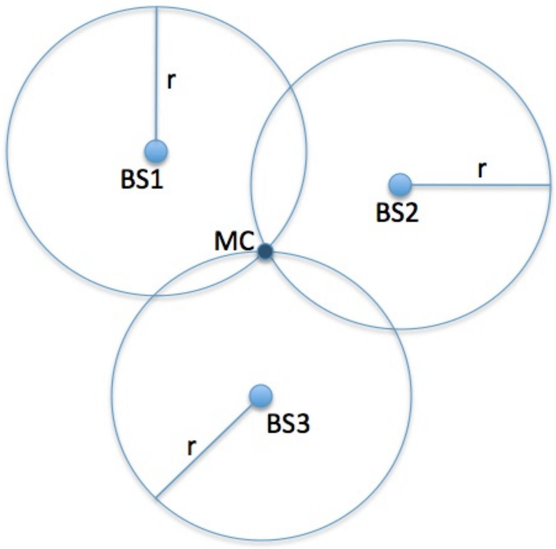


Figure 3.2: Trilateration

accuracy of the RSSI reading is not specified" [28]. Therefore, it is not possible to map the distance to the received signal strength directly. Instead an extensive measurement of the signal strength in the localisation environment is needed to explore the influences of obstacles like walls, doors and furnitures. Additionally a "decisive drawback of this method is a mandatory re-training phase, once the environment has been subject to major changes, like moved furniture or a rearrangement of an assembly line" [18].

Nevertheless the received signal strength was subject of many research papers and localisation projects as shown in [11]. It was the first approach for the localisation of mobile clients in wireless networks and has still relevance for special applications. Because of its simplicity and availability in every WLAN device it could be a good solution for applications which do not require a high localisation accuracy for example if an exact to the room localisation is needed.

Measuring by using the Signal Propagation Delay

The second group of approaches for measuring the distance between a client and the reference positions is to time the signal propagation delay. The electromagnetic signals of a wireless network propagate with nearly speed of light. Equation 3.1 is the definition of the velocity of propagation of an electromagnetic wave. The resulting speed by adding the parameters for an IEEE 802.11 WLAN ($\lambda = 0.12m$, $f = 2.4GHz$) is depicted in 3.2. The difference compared to the speed of light in vacuum (illustrated by equation 3.3) is marginal. Because of the the relatively small difference (0.299 m/ns when assumed light speed to 0.288 m/ns when taking the characteristics of IEEE 802.11 WLANs into account) often the speed of light is assumed as propagation speed for wireless networks. Furthermore, this visualizes the accuracy which is needed by using the described metric for localisation applications.

$$v = \lambda * f \qquad \text{(VoP}^3\text{of an electromagnetic wave)} \tag{3.1}$$
$$v = 2.88 * 10^8 m/s \qquad \text{(VoP in a 802.11 WLAN)} \tag{3.2}$$
$$c = 2.99 * 10^8 m/s \qquad \text{(speed of light in vacuum)} \tag{3.3}$$

In the following three common methods using the signal propagation delay for calculating the distances between the mobile client and the base stations are introduced.

Time of Arrival (TOA) The *Time of Arrival* is a measurement of the absolute time which one data frame needs to be transmitted from the source to the sink (see equation 3.4 [21]). This means from the base station to the mobile client or vice versa. By applying this technique a time-stamping of the outgoing and incoming data is required. Picture 3.3 depicts a time-stamping at the sink and the source which doesn't have to be realised by adding it directly to the frame. Instead it could be implemented just by saving a timestamp when the frame reached the sink. For avoiding to change the frame by adding a timestamp at the source too, it could also be saved and sent to the source in a follow up message like described in 3.3.2 and 3.3.3. Nevertheless, both devices, client and base station, have to be able to generate timestamps exact to the nanosecond which, as described in 3.2.1, is a tough task. Furthermore, the clocks of all involved devices (at least three base stations and one client) have to be synchronised [8, 11].

$$t_n = t_0 + \frac{d_n}{v} \qquad \text{(TOA for BS n)} \tag{3.4}$$
$$d_n = (t_n - t_0) * v \qquad \text{(distance MC to BS n)} \tag{3.5}$$

where:
t_n = time of flight between the client and the base station n
t_0 = time when beginning the measurement
d_n = distance between the client and the base station n
v = velocity of propagation

Round Trip Time (RTT) The *Round Trip Time* is the period, the signal needs travelling from the source to the sink and back. This has to be done with at least three base stations. Therefore, it has to be taken into account that the client is moving during the three measurements. This will distort the measurement, especially if the object that has to be located is able to move very fast. Because only one clock is used to determine the roundtrip time no synchronization between the different devices is needed. In return another problem occurs: Additionally to the time of flight to the mobile client and back to the base station, it needs a certain time for the client to send back the received frame. This period, which is used for processing the data at the sink before responding to the source has to be considered when calculating the distance to the base station [8]. 3.6 to 3.8 depicts the RTT by using the TOA equation 3.4. The error which would be added to the measurement is ignored and not part of the equation.

[3]Velocity of Propagation

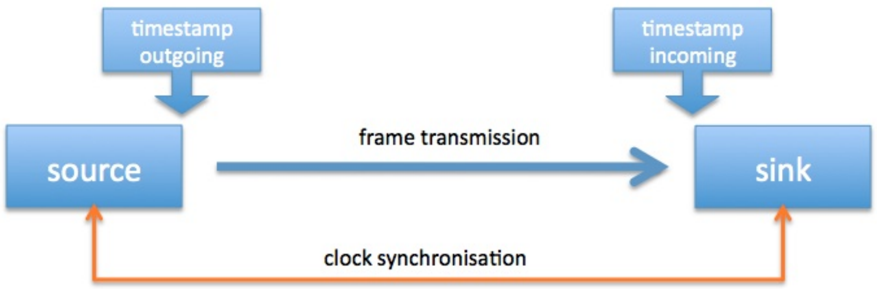


Figure 3.3: TOA concept

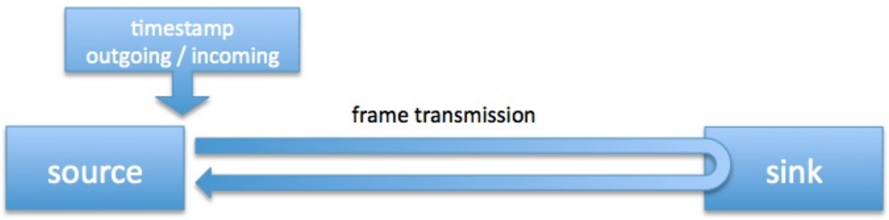


Figure 3.4: RTT concept

$$RTT = 2 * TOA \qquad \text{(RTT from TOA)} \tag{3.6}$$

$$t_{RTT} = t_{TOA} * 2 + t_{Processing} \qquad \text{(RTT with MC processing time)} \tag{3.7}$$

$$t_{RTT_n} = (t_0 + \frac{d_n}{v}) * 2 + t_{Processing} \tag{3.8}$$

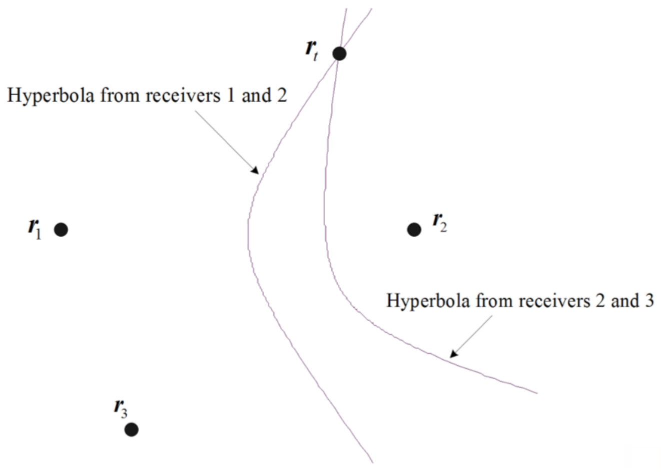
where:
$\mathrm{RTT_n}$ = Round Trip Time between the client and base station n

The transformation for calculating the distance from the mobile client to base station n is depicted in 3.9 to 3.10. Instead of absolute times (e.g. timestamps) relative times are used. Therefore, $\mathrm{t_0}$ can be dropped.

$$t_{RTT_n} = (\frac{d_n}{v}) * 2 + t_{Processing} \qquad \text{(RTT without } \mathrm{t_0}\text{)} \tag{3.9}$$

$$d_n = [\frac{t_{RTT_n} - t_{Processing}}{2}] * v \qquad \text{(distance calculation)} \tag{3.10}$$

Time Difference of Arrival (TDoA) Another method for determining the distance from the mobile client to the access points is the *Time Difference of Arrival* method. The idea is that the mobile client is sending a broadcast frame which is received by at least three base stations (for a 2D localisation) at different times. The base stations are saving the time of arrival and a central entity (i.e. a localisation server) computes the time differences between



r_t
Hyperbola from receivers 1 and 2
r_1
r_2
Hyperbola from receivers 2 and 3
r_3

Figure 3.5: TDoA localisation, taken from [10]

the access points, when the frame arrived at each of them. The time difference of two base stations and their locations are used to calculate a hyperbolic curve on which the mobile client is located. For evaluating the exact position the time differences between the two previously mentioned and the third base station is needed. This is illustrated in 3.5 where $\mathrm{r}_{1\text{-}3}$ are the base stations and r_t is the mobile client.

As depicted in 3.6 the clocks of the access points have to be synchronised with each other. But since only the time differences are measured no clock synchronisation with the mobile client is needed [7, 8, 11].

For calculating the time difference of arrival the time of arrival can be used, too. Equations 3.11 to 3.13 are depicting the transformation [12, 21, 26].

$$TDOA = TOA_1 - TOA_2 \tag{3.11}$$
$$t_1 - t_2 = \frac{d_1 - d_2}{v} \tag{3.12}$$
$$v * (t_1 - t_2) = d_1 - d_2 \tag{3.13}$$

Figure 3.7 depicts the triangle which has to be calculated to determine the mobile client's (MC) distance to one of the base stations (BS). By knowing the position of the base station from the floor, in which it is mounted on the wall, one side of the triangle is already known. Furthermore, the figure depicts that, in this case, only a 2D measurement is done and it is assumed that the client is located on the floor. Now, the distances in equation 3.13 can be replaced by geometric notation. By further using the pythagorean theorem equation 3.14 is received [12]:

$$v * (t_1 - t_2) = \sqrt{(x - x_1)^2 + (y - y_1)^2} - \sqrt{(x - x_2)^2 + (y - y_2)^2} \tag{3.14}$$

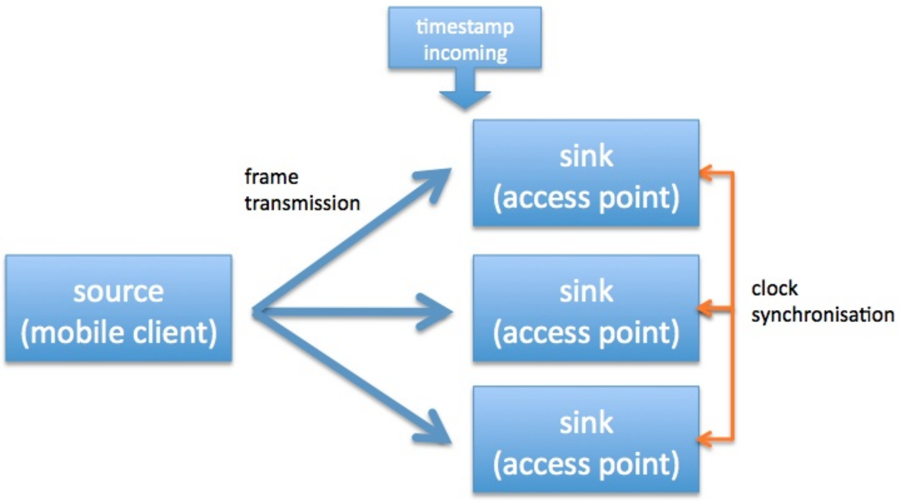


Figure 3.6: TDoA concept

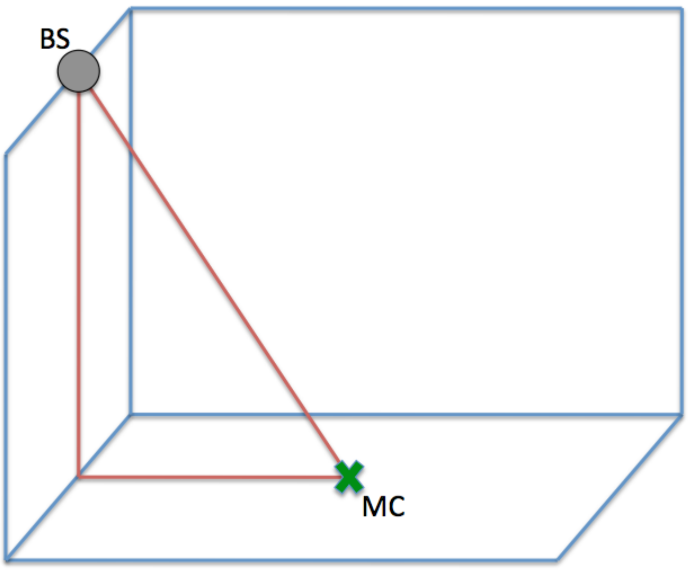


Figure 3.7: Geometric calculation

By having three base stations we obtain the following equations [12]:

$$v * (t_1 - t_2) = \sqrt{(x - x_1)^2 + (y - y_1)^2} - \sqrt{(x - x_2)^2 + (y - y_2)^2} \tag{3.15}$$

$$v * (t_2 - t_3) = \sqrt{(x - x_2)^2 + (y - y_2)^2} - \sqrt{(x - x_3)^2 + (y - y_3)^2} \tag{3.16}$$

$$v * (t_3 - t_1) = \sqrt{(x - x_3)^2 + (y - y_3)^2} - \sqrt{(x - x_1)^2 + (y - y_1)^2} \tag{3.17}$$

3.2.2 Angulation Techniques

In addition to the trilateration the *angulation* is the second measuring principle for locating mobile clients in a wireless network. The Angle of Arrival (AOA) which is depicted in 3.8 is one method which uses the intersection of angle direction lines radiant from the base stations (BS) to identify the location of the mobile client (MC). To extend the shown 2-D positioning example for a 3-D measurement a third base station is required. The advantage of an angulation is that "no time synchronization between measuring units is required. The disadvantages include relatively large and complex hardware requirement(s), and location estimate degradation as the target moves farther from the measuring units" [11].

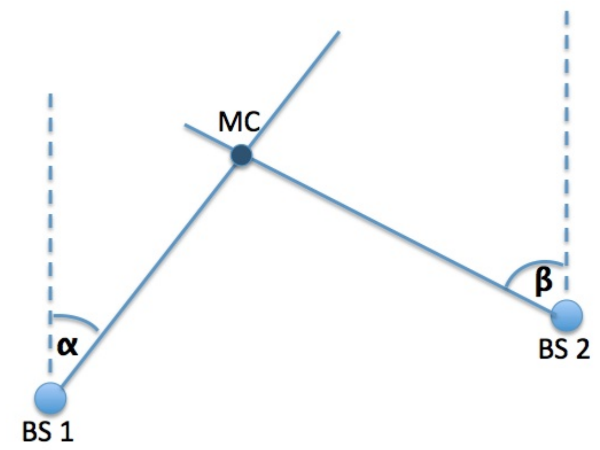


Figure 3.8: AOA concept, adapted from [11]

3.3 Signal Propagation Delay Measurement

The signal propagation delay measurement is the metering of the time the data needs traveling from the sink to the source or vice versa, depending on the used measuring principle (see section 3.2). This process is dependent on the digital clock, the synchronization of the clocks as well as some other influences discussed below.

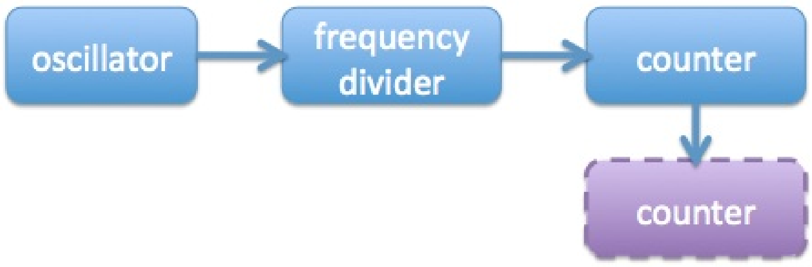


Figure 3.9: Parts of a digital clock

3.3.1 Digital clocks

Digital clocks are a common part used in watches or other electronics. 3.9 roughly depicts the parts of a digital clock and its functional principle. As shown, an oscillator, mostly of quartz crystal, is used to create an electric signal of a certain frequency. Quartz is a piezoelectrical material which is vibrating when a voltage is applied. The frequency generated by this kind of oscillators is very stable and, by cutting the quartz crystal in different ways, almost every frequency can be created [19]. In digital clocks the frequency is usually 32,786 Hz which is used to generate one pulse per second by using a down streamed frequency divider of 2^{15}. Several counters are used for counting to minutes, hours and so on.

A frequency of 32,786 Hz enables a resolution of milliseconds. But due to our requirement for a measurement exact to a nanosecond which means a billion of a second, this isn't enough. By keeping in the binary system at least a frequency of 1.074 GHz (2^{30}) which corresponds to a period $T = 931.3ps$ is needed to realize a nanosecond exact measurement.

3.3.2 Clock Synchronisation via IEEE 1588

As mentioned in chapter 3.2.1 clock synchronisation is an inevitable part of the localisation when using certain methods which are based on measuring the signal propagation delay. One method for synchronising clocks over a network like Ethernet is the IEEE[4] 1588 standard (also called PTP[5]) for a precision clock synchronisation protocol for network measurement and control systems. According to IEEE "existing protocols for clock synchronization are not optimum for these applications. For example, the Network Time Protocol (NTP) targets large distributed computing systems with millisecond synchronization requirements" [27]. In contrast the IEEE 1588 permits synchronisation better than 1 ns which should be reached by generating a master-slave architecture where "all clocks ultimately derive their time from a clock known as the grandmaster clock" [27]. 3.10 depicts an example hierarchy: Clock one is the grandmaster which is connected as a master to a slave port of boundary clock one. The boundary clocks are distributing the grandmasters time like a switch. The master ports of boundary clock one are connected to an ordinary clock as well as another boundary clock which is connected by its slave port and again distributes the time via its master ports.

The basic synchronisation message exchange (shown in 3.11) proceeds as follows: The master sends a synchronisation packet to the slave and notes the time (t_1) it was sent. The slave notes the time the packet arrived (t_2). The master has two possibilities to inform the

[4]Institute of Electrical and Electronics Engineers
[5]Precision Time Protocol

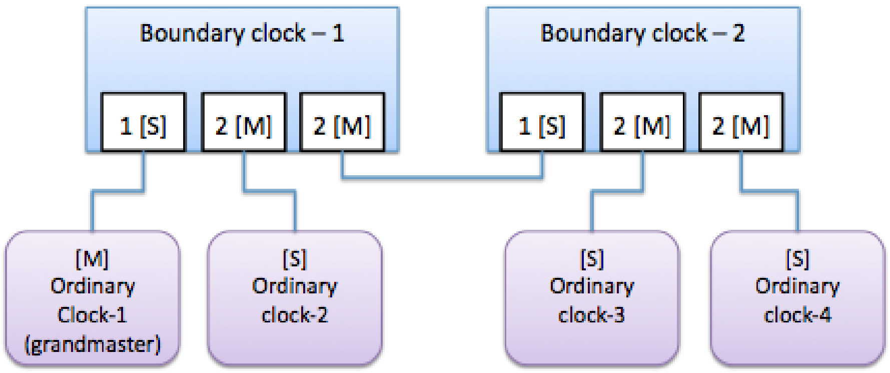


Figure 3.10: Clock hierarchy in IEEE 1588, adapted from [27]

slave about t_1:

- Embedding t_1 in the synchronisation message which requires a high effort to avoid any delays during the time-stamping process.

- Embedding t_1 in a follow up message sent directly after the synchronisation message as shown in 3.11.

Now the slave knows the time the packet was sent by the master as well as the time it arrived. By assuming that the master-to-slave and slave-to-master propagation times are equal another message exchange takes place for calculating the offset between the slaves clock respective to the masters one. For this purpose the slave sends a delay request message to the master and notices the outgoing time (t_3). The master sends the timestamp of the incoming of the delay request (t_4) as a delay response back to the slave. Now the slave knows about all four times and is able to process the respective calculations.

The timestamps discussed above should be created at the transition of the node and the network. The nearer to the network the time stamp is created the smaller are the timing errors. These emerge due to the time lapse from the creation of the timestamp till the data has traversed the upper layers to the actual network connection. A solution is to use a hardware assist between the MAC[6]- and the physical layer (figure 3.12). In [16] Patrick Loschmidt et al. analysed the limits of synchronisation accuracy using hardware support. While using hardware support to eliminate the jitter generated by the higher layers like IP[7] and UDP[8], they identified further jitter sources on and below the physical layer [16] which are introduced in chapter 3.3.5.

This brief introduction to the IEEE 1588 clock synchronization is a basis for further discussions in this document. For detailed and further information the standard [27] should be consulted.

[6]Medium Access Control
[7]Internet Protocol
[8]User Datagram Protocol

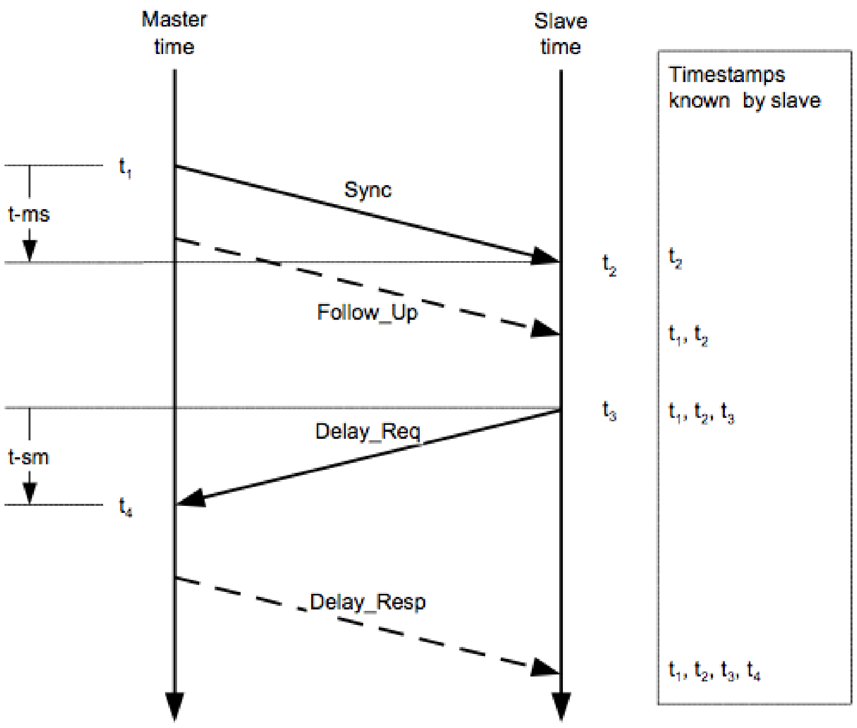


Figure 3.11: Basic synchronisation message exchange, taken from [27]

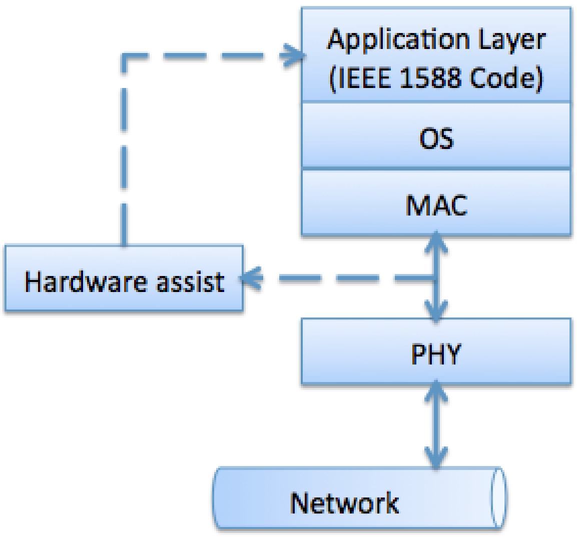


Figure 3.12: Timestamp generation, adapted from [27]

3.3.3 Timestamping

Creating timestamps is not only needed for the clock synchronisation as described in chapter 3.3.2. Another field of application is the measurement of the time of flight for determining the distance from the mobile client to the base stations as described in section 3.2.1. In both cases the timestamps have to be as precise as possible, in our case minimum exact to the nanosecond.

As already mentioned a timestamp can be added to the actual frame which has to be timestamped or the timestamp can be saved as soon the frame passes a certain event. By using the second option the incoming and outgoing frames are only monitored, not changed. Here it is important that the frames ID is recorded next to the actual time. Otherwise it is not possible to refer the recorded times to the frame and therefore to its origin anymore. If the timestamp should be added directly to the frame there are some things to be considered, too:

1. The length of some fields like the header of the data are varying. Hence, "either the assumption of a fixed position must hold or the hardware unit must be able to interpret all necessary header information to determine the correct position in the frame" [22]

2. The insertion of a timestamp involves the recalculation of the CRC[9] which may have been done before. For some protocols such as UDP[10] a recalculation is not possible or is that time consuming that the processing time would corrupt the measurement. Though, a benefit of UDP is the possibility to simply set the checksum to zero. This suspends the actual purpose of the CRC and furthermore doesn't work with every protocol. [22]

Next to the place where the timestamp is saved different methods where the timestamp is recorded exists: The first option for recording timestamps is a software solution and the second one is using a hardware support near to the PHY. This second solution was described before and shown in figure 3.12: a hardware support between the MAC and the PHY that reduces the jitter from the upper layers. According to [22] the IEEE defines the Media Independent Interface (MII) as the interface closest to the medium. Therefore, it is stated as the optimal access point if COTS[11] products should be used. Further it's mentioned that "cost reductions and size issues lead to highly integrated, combined PHY/MAC solutions" [22] and that in this case "the required physical signals for a timestamper are no longer available on the MII level" [22]. The other option is creating timestamps on the software level inside the device driver. "When a packet is sent or received, the hardware notifies the driver with a hardware interrupt. The driver masks the interrupt, takes the timestamp using the *getnstimeofday* interface of the kernel and then instigates a corresponding interrupt handling routine" [3].

3.3.4 Environmental considerations

Regarding the environmental considerations we differentiate two cases: Line of sight and non line of sight environments, both explained below.

[9]Cyclic Redundancy Check
[10]User Datagram Protocol
[11]Commercial off the shelf

Line of Sight (LOS)

A line of sight environment exists if there is a line of sight between the mobile client and the base stations used to locate the client. Like explained in the next chapter this case is rare in normal life applications. Nevertheless it is widely mentioned in contemporary literature [11,14,15] and sometimes also used as a reference environment. The angle of arrival method explained in 3.2.2 for example is dependent on a line of sight environment. And the requirements pointed out for this thesis are based on a line of sight environment, too. Hence, every room should be equipped with a minimum of three base stations to avoid heavy obstacles like walls between the base station and the mobile client. Nevertheless, the most applications will require a localisation system able to work in a non line of sight environment which is explained below. At least smaller obstacles like persons or furnitures which could hinder a direct line of sight exists in every room.

Non Line of Sight (NLOS)

A non line of sight environment is most likely to be encountered when designing an indoor localisation system for a real world application. Most rooms and buildings are full of obstacles like tables, lockers, people and so on, which obstruct a clear line of sight between the mobile client and the three or more base stations needed for the localisation. Since, like shown in the previous chapters, no off-the-shelf antennas could be used for most of the localisation purposes, one will try to use as less costly special build ones as possible. Hence, cost concerns will lead to the wish to cover as much space (also different rooms) with as less base stations as possible. Therefore a precise localisation in non line of sight environments should be the future task to be solved.

3.3.5 Influences on the signal propagation delay measurement

This section summarises the influences on the signal propagation delay measurement discussed previously and additionally introduces some new points.

The first and essential influence is "the clock rate (resolution of the time stamp) and the stability of the oscillator used to generate the nodes internal clock" [16]. 3.3 illustrates that the available accuracy of widely used crystal oscillators doesn't fit the needs for a high precision time measurement exact to the nanosecond.

Second is the protocol for synchronising the different clocks to be able to measure the propagation delay for metering the distance from the base stations to the mobile client as illustrated in chapter 3.2.1. We have seen that the accuracy using the network time protocol (NTP) isn't sufficient, hence the IEEE 1588 protocol was introduced. However, also using the IEEE 1588 some points have to be considered to reduce or eliminate the jitter:

1. The creation of the timestamp should take place at the lowest possible level, if possible directly at the transition to the physical layer. As mentioned in 3.3.2 this helps to reduce the timing error.

2. [16] evaluates the impact of the synchronisation interval on the accuracy of the clock synchronisation. It is shown that a diminished interval leads to a higher standard deviation of the clocks accuracy. This is obvious and depends on the oscillators stability. If an oscillator for example deviates 4 nanoseconds from its target every 2 seconds it

has to be synchronised more often than an oscillator which deviates only 2 nanoseconds every 2 seconds.

3. The master-slave architecture of the PTP obviously entails a single point of failure, the grandmaster clock. [9] treats the potential risk of a loose of accuracy while determining a new grandmaster clock if the original one fails. However in our case a continuous measuring is secondary, hence this point is neglectable. But a measurement affected by a clock failure should be recognized and repeated after the operating conditions are re-established.

4. Additionally [4] mentions problems when trying to implement a clock synchronisation for very huge networks: "However, in case of a large-scale network it is not likely to have the possibility to distribute a single clock frequency throughout the whole system. Consequently, principles which can be expanded to large-scale networks need to be found" [4]. Therefore, before implementing a clock synchronisation it has to be checked if the clock synchronisation domain can be devided for example building-wise or even floor-wise if the building is very huge. This entails possible synchronisation problems at the outskirts of two domains when base stations of both domains have to be used. But in many cases this interference neglectable.

A third aspect affecting the signal propagation delay measurement occurs in non line of sight environments (which are described in chapter 3.3.4). This so called multipath-problem arises when the signal from the mobile client is reaching the receiving base station on different paths. Causes of multipath are reflections of the signal by obstacles in the environment, like people or furnitures. Furthermore, the signal also can be reflected by the rooms floors and ceilings. This leads to a mitigation of the signal or even to its loss, if the signal strength comes below the threshold the base station is able to recognize. On the other hand the reflections can directly influence the measurement because a reflected signal has a longer propagation time from the source to the sink than a signal which travels the direct way [15]. [15] proposed a method to mitigate this problem by analysing the multipath condition of the channel and choosing an appropriate distance estimation algorithm for the measurement.

Furthermore, an influence of the transceiver amplification on the measurement was discovered in [8]. It has been shown that the used boards shown a non-linear, amplification dependent group delay. "The higher the gain of the amplifier, the higher the group delay of the signal" [8]. Group delays occur when passing a signal through a medium or a device and in this case generate, as depicted in 3.13 and 3.14, measurement errors in the nanosecond range. This error which will differ depending on the used hardware has to be compensated for example by adding a function to the device software which is subtracting an amount of time depending on the transceivers attenuation.

3.4 Localisation supporting algorithms

Besides a well planned and precise hardware system, algorithms can be used to improve the accuracy and form the localisation system.

The supporting possibilities of algorithms range from simply tracking the last positions of a moving object to be able to assure the upcoming measurement to the use of Kalman filters and genetic algorithms. The first example could be used at our hospital use case. By assuming that the location of a patient in the hospital will be established every 3 seconds and by saving

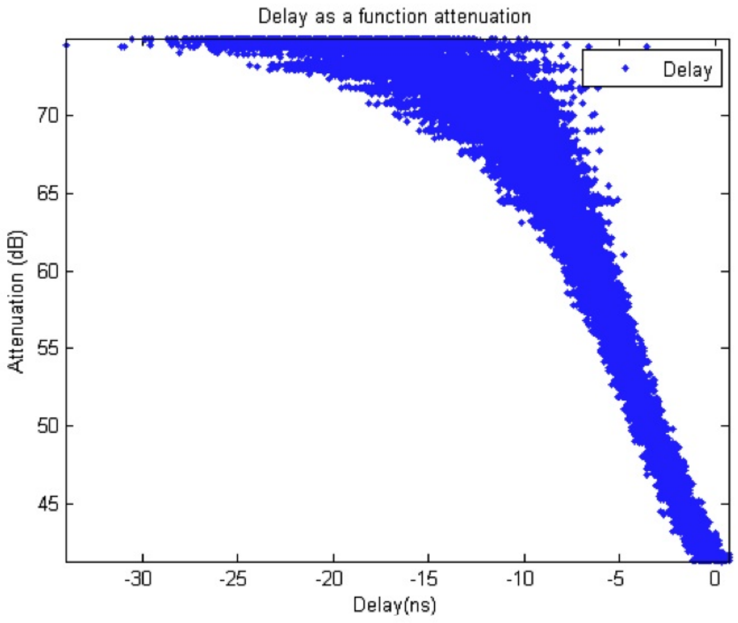


Figure 3.13: Group delay with rising attenuation, taken from [8]

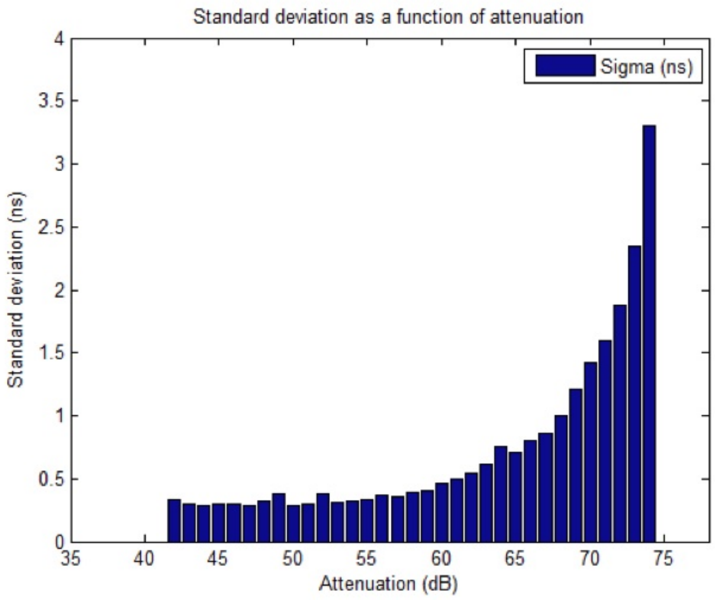


Figure 3.14: Group delay standard deviation, taken from [8]

the last few locations of the patient we can try to predict the upcoming measurement. If a patient moves with about 4 km per hour he / she will be located within a radius of 3.3 meters from his/her last position. Of course this alone isn't a very precise prediction and it will be stretched to its limits if the patient is for example using an elevator. In this case the algorithm would report an error. But if the upcoming measurements are logically correct again, we can drop the wrong prediction. On the other hand, if the reported locations differ all the time we can assume an error like a reflection of the signal occurred. However, by adding information like maps or more precise moving speeds to the algorithms, the accuracy will enhance and can be a helpful add on.

In [13] Luis Mengual et al. introduced a software support for RSSI based locations. They proposed a localisation by using fingerprints of the environment. These are measurements of the the signal strength of the different base stations at several points in the environment to build a so called radio-map. When establishing a localisation the mobile clients actual measurements are compared to the radio-map by using a neural network which determines the most certain position of the client by looking at the existing data. In [17] Qiuxia Chen et al. argued that "obviously, these methods do not perform well because RSSs change over time due to the dynamically changing environment." Non-changing environments are not common in real world scenarios, which reduces the importance of radio-map based RSSI localisation methods.

In another example which was already mentioned in chapter 3.3.5 algorithms are used to improve the localisation accuracy in non-line-of sight environments. The method proposed in [15] reduces the error originated by multipath by analysing the chanel conditions and then choosing an appropriate distance estimation algorithm.

These are only some examples for possibilities to increase the location accuracy by deploying appropriate algorithms. Summarizing it has to be stated that a good localisation system always contains a well designed software next to its hardware.

3.5 Conclusion

As will be seen in the next chapter some intensively described methods of this chapter aren't feasible to be used for the defined use case. But describing the state-of-the-art localisation techniques and their characteristics was an essential part for being able to reconstruct the arguments for banning certain methods and favor others during the evaluation in this next chapter. Furthermore, they are important when having another use case at hand. And since this work is supposed to give a general overview to techniques and problems involved in wireless mobile client localisation this pleadings was inevitable.

Chapter 4

Evaluation of the State of the Art

The following chapter evaluates the state of the art techniques discussed in chapter 3 and constitutes the basis for the development of a system blueprint, described in chapter 5. It compares the different methods regarding usability and efficiency by paying attention to the defined use case and its requirements.

4.1 Evaluation of the Localisation Strategies

Because of the already mentioned legal considerations the mobile client involved localisation has to be preferred. A possible localisation of every person who is around the antennas reception area will most certainly lead to legal problems. Such as that employers are not allowed to monitor their staff especially not during their brakes. Moreover, the guests visiting friends and family could have doubts about to expose where they are and whom they are visiting. And at least there are the patients who don't need or even don't want to use the service of being detected in an alarm case. Hence the mobile client involved localisation is the right strategy to choose.

However, this use case enables another opportunity which is not practical in a lot of other cases. Beside the use of a mobile phone application on the client side as introduced in the requirement specification of chapter 2, a mobile client device especially developed for this case can also be used. Because a hospital area as well as the targeted user group is very delimited it could be worth to develop a mobile device which is more convenient regarding a time-stamping of the frames when using the TDoA[1] approach or regarding a short processing of the data when responding to the base station as postulated when using the RTT[2] method. Therefore the costs of developing such a device as well as the resulting benefits have to be analysed to be able to make a decision about that.

4.2 Evaluation of the Measuring Principles

The evaluation of the measuring principles contains of the evaluation itself as well as a previous definition of possible evaluation criteria.

[1]Time Difference of Arrival

[2]Round Trip Time

4.2.1 Definition of the Evaluation Criteria

For evaluating the methods described in the previous chapters some criteria based on the use case requirements has to be defined. Georg Gaderer et al. for example used the dimensions accuracy, range and power awareness for comparing the different localisation approaches [8]. The accuracy of the overall measurement system is a criteria here, too. But because of the line of sight requirement every room has to be equipped with base stations the range most certainly will not be a problem and can be dropped as a criteria in this case. The power awareness at the mobile client side could be an important criteria, too. If a localization is only needed in alarm cases the power consumption will be low. Emergencies will be rare and the rest of the time only some regularly status messages (so called "I am alive messages") have to be sent to be sure that the client is still working properly. But if, for some reason a continuous localisation is required the power consumption of the mobile client will become an important criteria. However, since not many implementations of the measuring principles exist, an evaluation of the power consumption is hard and only theoretical. Therefore, this criteria will not be used here either.

Further dimensions that should be used here for the upcoming evaluation are the costs and the simplicity of the system. The costs include the price of the hardware as well as for the operation and possible extensions of the localization system. Simplicity contains different points like the use of common techniques where possible, as well as a reliability and fault tolerance through simplicity. Therefore we have the following criteria for rating the measuring principles as well as the overall localisation system:

- Accuracy
- Costs
- Simplicity

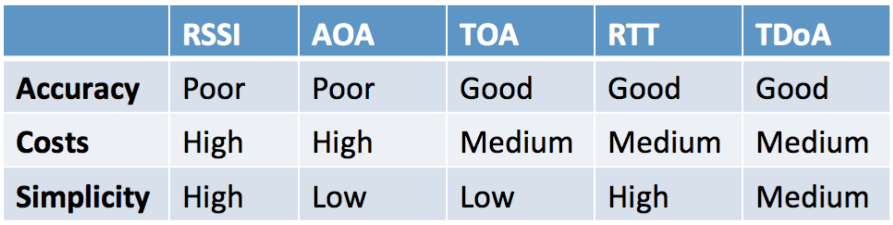
Furthermore, the before mentioned possibility of building a special client because of the limited user group should be concerned.

4.2.2 Evaluation

As discussed before, establishing an angulation system like the described angle of arrival method requires extensive effort especially regarding the needed hardware. Additionally the reachable accuracy is relatively low, especially when the distances between base station and mobile client get larger. Therefore, a lateration principle is the preferable to use.

A hospital is a very changing environment. Beds as well as lockers are moved as needed and a lot of people are walking around all the time. Rooms can be empty and one hour later crowded of visiting family members. In places like these RSSI[3] based measuring principles will reach a very poor localisation accuracy and still a lot of research is needed to mitigate those problems. The once created radio maps which are the basis for an RSSI based method will be useless very soon. Keeping the maps up to date would require extensive and very costly re-measurements. Dimitrios Lymberopoulos et al., who analysed the impact of the base stations antennas and their orientation on the RSSI measurement in IEEE 802.15.4 networks, concluded their paper [5] as followes: "Our results and experience from this work show that signal strength localization will work in specially instrumented scenarios. In other scenarios

[3]Received Signal Strength Indicator



	RSSI	AOA	TOA	RTT	TDoA
Accuracy	Poor	Poor	Good	Good	Good
Costs	High	High	Medium	Medium	Medium
Simplicity	High	Low	Low	High	Medium

Figure 4.1: Comparison of the measuring principles

and 3D deployments, signal strength localization remains an extremely challenging task. Statistical techniques and specific deployment scenarios will mitigate some of these challenges. However, the large amount of characterization needed will make the use of signal strength approaches with low power radios practically impossible" [5]. Additionally, by looking at their results, Francesco Potorti et al. finished their paper about RSSI in-room localisation with the following sentence: "This suggests that much investigation is needed for single-room RSSI localisation to attain any useful performance" [6]. Hence, the distance of the mobile client to the base stations should be measured by a signal propagation delay method. The three methods left which are in line to be used are the time of arrival (TOA), the round trip time (RTT) and the time difference of arrival (TDoA) measurement. As described before, TOA requires a time synchronisation between the mobile clients and all of the base stations clocks. The synchronization between the base stations is also needed when using TDoA and can be done via IEEE1588 over the cabled network. But when the mobile clients clock has to be synchronised too this has to be done over wireless LAN[4]. Because of the relatively unefficient medium access control used in WLANs and the resulting timing uncertainties this is a much more complicated task. Hence, time of arrival isn't the preferred measuring principle to use. Now, one main characteristic is differentiating the two methods which are still in line: When using the time difference of arrival method a clock synchronisation between the base stations is needed for measuring the difference in time between the three or more base stations in which the packets are arriving. This is a possible but challenging task especially when a high number of base stations which are distributed over a wide area have to be synchronised. In addition, the synchronisation system is another single point of failure since the whole localisation system depends on it.

Both methods are possible candidates. The TDoA is the best method for the more general cases where a lot of different devices use the system and the main tasks should be shifted to the base stations. And the RTT method is the best for applications where a small and very specific user group exists and where it is worth to develop a mobile client with real time characteristics if no one is available off the shelf. And by having a closer look at the use case definition we can see that the localisation is not needed for everyone in the hospital but only for special patients. Therefore, the localisation only has to work with a special mobile client which is carried around by the patients instead of a broad range of devices like cell phones and computers.

Figure 4.1 is based on the points discussed before and makes the different measuring principles easy comparable. Regarding this evaluation the round trip time measurement is the preferred method to use.

[4]Local Area Network

Chapter 5

System Definition

The target of the system definition chapter is to deliver a blueprint for a hospital patient localisation system independent of any hardware or specific implementation. It is resting upon the evaluation of the state of the art done previously and ensues the proposed methods and techniques.

5.1 Comprehensive System Design

[24] proposes a functional block diagram for a wireless geolocation system which is depicted in 5.1 It is also adaptable to our case and contains a number of sensing devices that measure metrics related to the position of a mobile client with respect to their positions which works as a known reference point. A positioning algorithm processes the metrics reported by the location sensing elements to calculate the position of the mobile client as well as a display system which illustrates this position. [24]

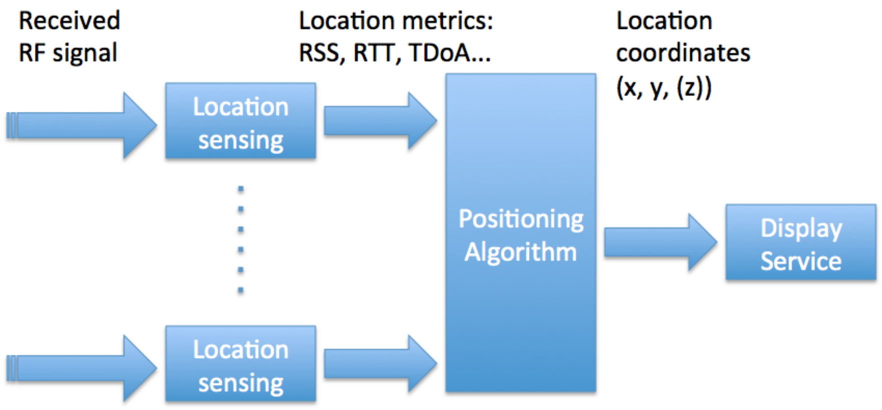


Figure 5.1: Functional block diagram, adapted from [24]

By looking at our case the location sensing devices are the base stations which are sending localisation frames to the mobile client and wait for their response. After they received the frames from the mobile client they forward them to the localisation server. The server collects the packets from the different base stations and processes the positioning algorithm. Moreover, it provides the data for the display system, which could operate on the localisation server

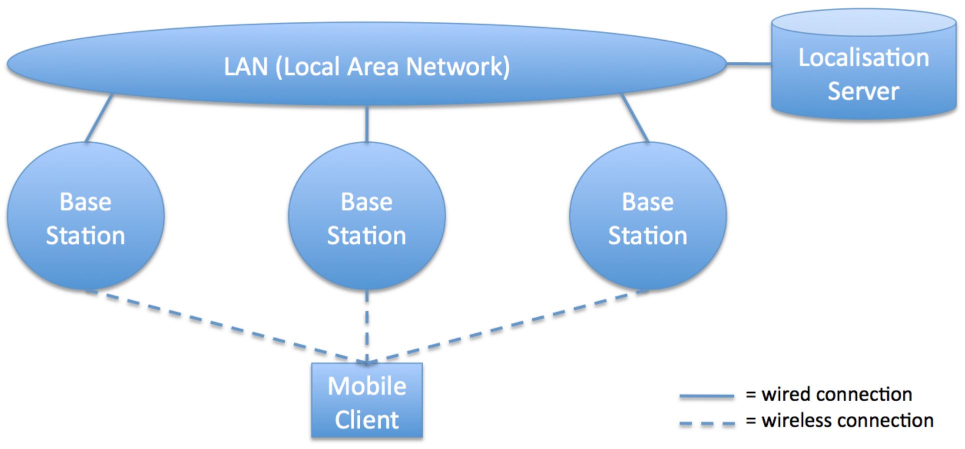


Figure 5.2: The localisation System

itself or another device which is connected to the network. Figure 5.2 depicts a diagram of the localisation system: The base stations as well as the localisation server are attached to the network by a wired connection. The mobile client communicates with the base stations over a wireless connection (e.g. IEEE 802.11g).

The localisation process is depicted in 5.3. It shows the interaction of the different components of the localisation system. Different starts of the process are possible. In diagram 5.3 the mobile client initiates the localisation. This could happen if the patient feels bad and pushes a button at his/her mobile client device. Afterwards the client sends a localisation / positioning request to the localisation server via the base station it is connected to. It possibly checks if the client is authorized for this application and sends a localisation order to all base stations near the one the mobile client is connected to. By letting the server work as controlling device and triggering the base stations on after another, collisions through the base stations can be avoided. Afterwards, every base station creates a localisation frame, timestamps it and sends it to the mobile client. The client processes the frame and redirects it back to the base station which timestamps it again and forwards it to the localisation server. The server waits for the response of all base stations and calculates the mobile clients‘ position based on the received data. Afterwards the server makes the position available to a display device which could be integrated into the localisation server or could operate as a piece of software installed on any computer in the network.

By knowing the first design draft of the system it is worth to take a look at the constraints the system has to fulfill. As mentioned in the use case definition, the accuracy requirement is that the mobile client device is to be located within a uncertainty of a two meter radius. By looking at the signals velocity of propagation which was was calculated 0.288 m/ns a maximum time uncertainty of 6.94 ns for the whole localisation system arises. When looking at the system design especially the localisation process 5.3 following devices have an impact on the measurement accuracy: the mobile client and the base stations. The localisation server may only compensate the failures occurred before by applying intelligent algorithms for calculating the clients position. The display function only shows the position to the user and has no influence on the systems accuracy. But a third influence which has not beed discussed in this chapter has a heavy impact, too. Meant is the localisation environment

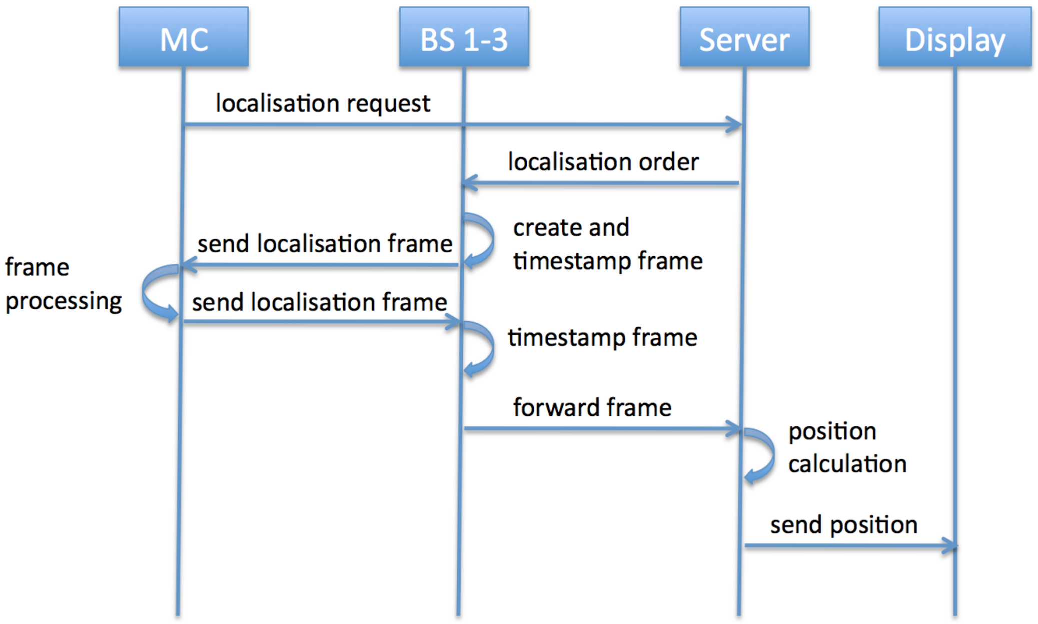


Figure 5.3: Sequence Diagram: Localisation Process

which can add uncertainties through multi-path (i.e. due reflections at obstacles) to the measurement. Without having any experience at the beginning the 6.94 ns buffer is to be divided as followed:

- Possible loss at the base station: 2ns
- Possible loss at the mobile client: 2ns
- Possible loss due to the environment: 2ns

Regarding the mobile client and the base stations the localization process 5.3 also unveils some important sub processes which are potential error sources. This is the creation and timestamping of the outgoing frame as well as the timestamping of the incoming frame at the base stations and the frame processing at the mobile client. Hence, a detailed look at the mobile client and the base station is provided in the following.

5.1.1 The Base Station

Next to the localisation the base station has to fulfill the tasks of a normal WLAN access point to avoid extra costs for installing an additional network for connecting wireless clients to the internal network or the internet. Hence, the base station contains of two functional blocks as depicted in 5.4. Consequently a standard access point design can be taken as a basis and subsequently extended and changed regarding the localisation requirements.

These localisation requirements are the following:

- To create a localisation frame containing a proper identification and the outgoing timestamp.

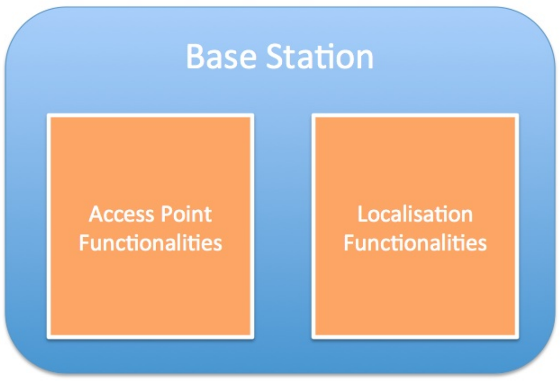


Figure 5.4: Base Station Functionalities

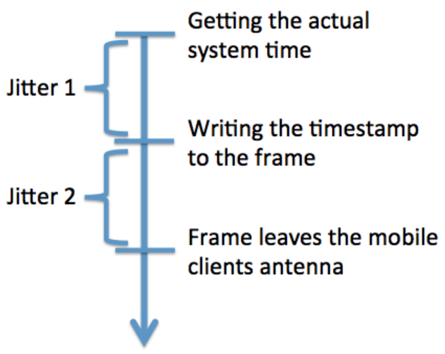

Getting the actual system time
Jitter 1
Writing the timestamp to the frame
Jitter 2
Frame leaves the mobile clients antenna

Figure 5.5: Jitters: Timeline

- To create the timestamps exact to the nanosecond
- To develop mechanism to scan incoming packets for localisation frames
- To provide an exact to the nanosecond time-stamping of the localisation frames directly after arriving at the base stations

In order that the localisation server is able to differentiate the localisation frames and store their information in the database properly, the frames need to carry an identification. This identification has to tell about the localisation process, which includes three frames (one from each of three base stations), the certain base station that was involved in the localisation as well as the client, that has been located.

Before the frame is sent to the mobile client the base station has to add a timestamp to it. As already discussed this has to be done as close to the physical layer as possible in order to generate a jitter as small as possible. This jitter is made up of two parts (see also figure 5.5): firstly the elapsed time between getting the actual time from the system and writing it as a timestamp to the frame. Second part is the elapsed time between the moment that the timestamp was added to the frame and the moment the frame leaves the base stations antenna and enters the medium to travel to the base station. These two periods add up to the jitter distorting the measurement. In addition, the timestamps resolution has to be exact to the nanosecond to reach the required accuracy.

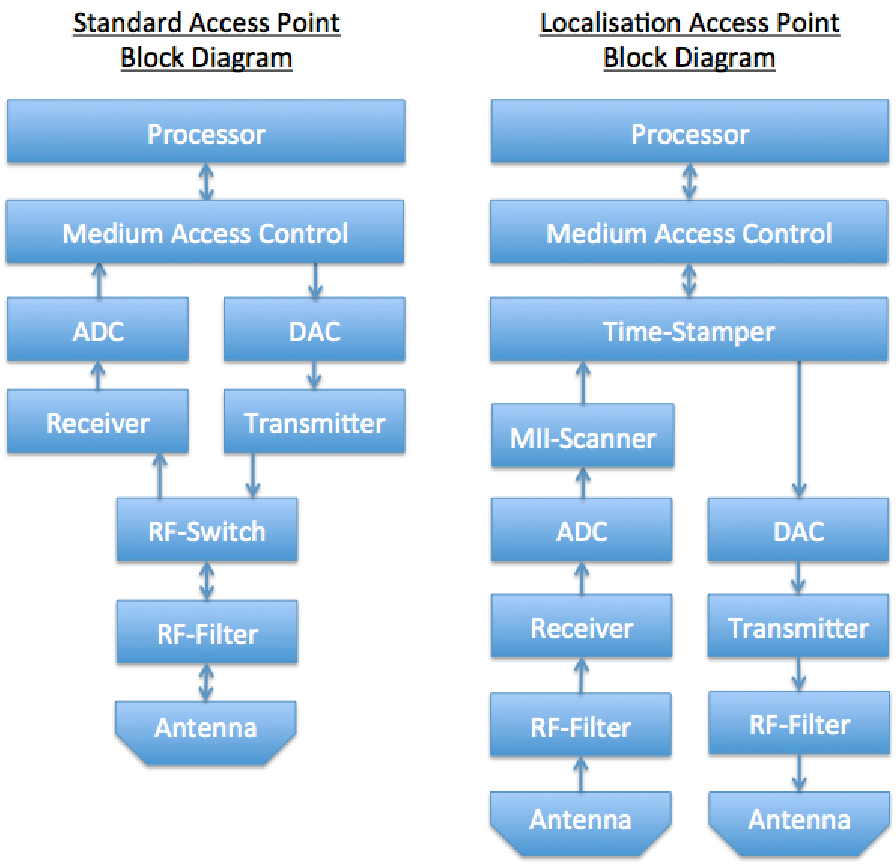


Figure 5.6: Block Diagram of a Standard- and a Localisation Base Station

After the frame has been returned to the base station by the mobile client, the base station has to be capable to recognise it as soon as possible to add the incoming timestamp to the frame. For keeping the jitter small again, the scanner also needs to be installed as near to the physical layer as possible. Further delays can be avoided by scanning the packets on the fly. The recognition can be eased by adding an identification at the beginning of the frame that marks it as a localisation frame. Then only the beginning has to be read to decide what kind of frame is present. Additionally, the different values of the frame should have fixed positions to be able to add the new data very quick. Without fixed positions, the delimiters between the single values have to be counted to find the right place for the certain value to be added.

Figure 5.6 depicts two block diagrams: one of a standard base station and a proposal for a localisation capable base station. Looking from the upper to the lower levels, the first difference is the additional Media Independent Interface (MII) scanner and the time-stamper of the localisation base station. If a normal packet is sent, it does not have to pass the time-stamper. And the MII-Scanner can be by-passed at all by outgoing frames. Incoming frames are scanned and, if a localisation frame is detected, it is handed to the time-stamper before reaching the medium access control.

The second difference is that the localisation base station has two antennas, one for receiving and one for transmitting. Hence, the receiving and transmitting parts can be split. This avoids that the system has to switch between receiving and transmitting mode, what would take time and result in another jitter.

5.1.2 The Mobile Client

The mobile clients task is to receive localisation frames sent by the base station and send them back again. But this very short description contains some complex points which are worth to have a deeper look at.

For the test system which has to be developed the client just has to send back every received frame as a broadcast whether it is important for its localisation or not. This means, no selection is done before processing the received data. For a system which has to be implemented at a hospital this isn't feasible any more. By resending every received frame the amount of data would be doubled with every mobile client used and the medium access would be more complicated because the medium is occupied more often. Hence, a localisation system would benefit by a more specific localisation request / answer. This means, that the client is reading the frame first and only processes the data if itself is the receiver.

As described before in 3.2.1 the processing time has to be subtracted from the round trip time to calculate the distance between the base station and the client. Hence, the processing time has to be predictable. This means that, next to the logical or functional correctness of the output, a timing correctness is necessary. According to [23] these are requirements of a real-time system.

Real Time Characteristics

As in the mobile clients case real time systems always require a logical correctness and a timing correctness. "The two notions of correctness may also be traded of against each other. The result is two broad categories of realtime systems: hard realtime systems and soft realtime systems. In hard realtime systems, timing correctness is critically important and may not be sacrificed for other gains" [23]. "In soft real time systems timing correctness is important but not critical" [23]. By looking at the mobile client and its before stated requirements it is critically important for the measurement that the processing is timing correct within the 2 ns buffer. Hence, the processing doesn't have to be as fast as possible but it has to be predictable within an uncertainty of 2 ns. In real time systems another distinction is made between time-driven and event-driven architectures. [23] The mobile client is an event-driven system as it reacts to an external event: the receiving of a frame at its antennas. This event starts an action, the frame processing, which leads to the timing and logical correctness based output. This functional description of the mobile client is depicted in 5.7 and is the basis for a more fine grained description.

As described before the test system does not need to distinguish between localisation frames or normal data frames send by other devices. Hence, any received frame will be treated as a localisation request. Later the differentiation has to be done by an on-the-fly interpreter which is preferably located between the PHY and the MAC. If a valid localisation frame is detected it will be forwarded for further processing in all other cases the frames are dropped.

Design and Frame Processing

The block diagram in figure 5.8 proposes a system design that takes the before discussed points into account. It is very similar to the base station block diagram with the difference

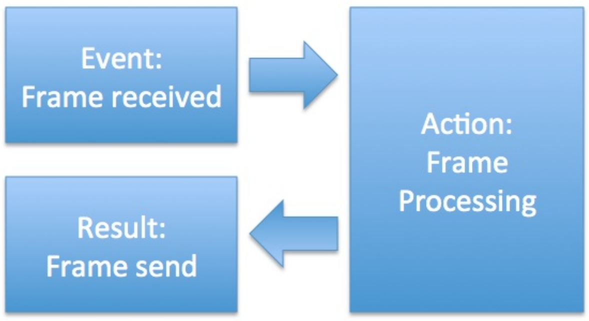


Figure 5.7: Real Time Characteristics

that no time-stamping functionality is needed and therefore does not exist. Due to the unpredictabilities by switching between send and receiving mode already discussed in 5.1.1 if only one antenna exists, those paths are separated here, too.

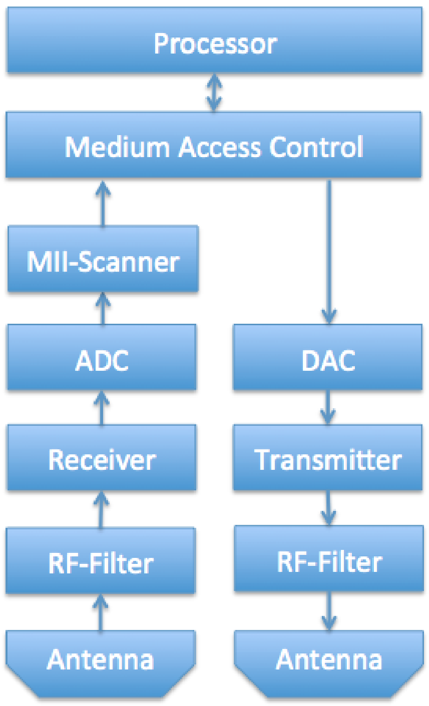


Figure 5.8: Mobile Client Block Diagram

As soon as a signal arrives at the receiving antenna it passes through the radio frequency (RF) filter and the receiver, where the analogue signal is processed before being converted into a digital one at the ADC (analogue digital converter). Afterwards, the MII (media independent interface) scanner looks for localisation frames to let them pass and drop all other data.

When it comes to the processor, one of the most unpredictable parts regarding the processing time begins. For keeping the uncertainty low, the choice of the programming language is a vital part. First opportunity is to use the most atomic one and write the software in assembler. This avoids any translation uncertainties caused by the compiler. Second option is to use a real-time capable high level language like Java Real Time System (Java RTS). Important differences to other languages are for example that it is assured that the garbage collection process does not start during a time critical process by giving the real time thread a higher priority than the garbage collector. Some other innovations of Java Real-Time System in comparison to Java are an advanced memory management which keeps enough memory free if the garbage collector is blocked by a critical thread as well as the elimination of a possible priority inversion.

After the frame has been processed and is ready to be sent back, the second critical part, the medium access control (MAC), has to be passed. The MAC checks if the medium is free. If this is not the case it waits for a certain time and tries again. This retry is an uncertainty which has to be avoided or managed. The most easy way is to drop the frame if it could not be sent at the first time and initiate a localisation retry. Second option could be to use a fixed wait time and add an information about the number of tries to the frame so that it could be factored into the distance calculation of the localisation server.

If the medium is free the frame passes the transmission path and is sent back to the base station.

5.1.3 The Localisation Server

The localisation server will be a piece of software installed on a computer connected to the network via cable. The sequence diagram 5.9 depicts the process which is started as soon as a frame is received by the server. The diagram is partitioned into three parts. The first one has to be done three times, i.e. one for every base station, and starts with sending the timestamped localisation frame to the localisation server. After it have been received, the program reads the frame, extracts the needed information and writes them into the database. The second part is the calculation of the mobile clients position and starts after all localisation frames from the base station has been received. Therefore, the program requests the needed localisation information from the database (e.g. the timestamps, the position of the base stations and so on), calculates the clients position based on those data and writes the calculated position back to the server. In step three, the program provides the position data to a display service that depicts the location on a map.

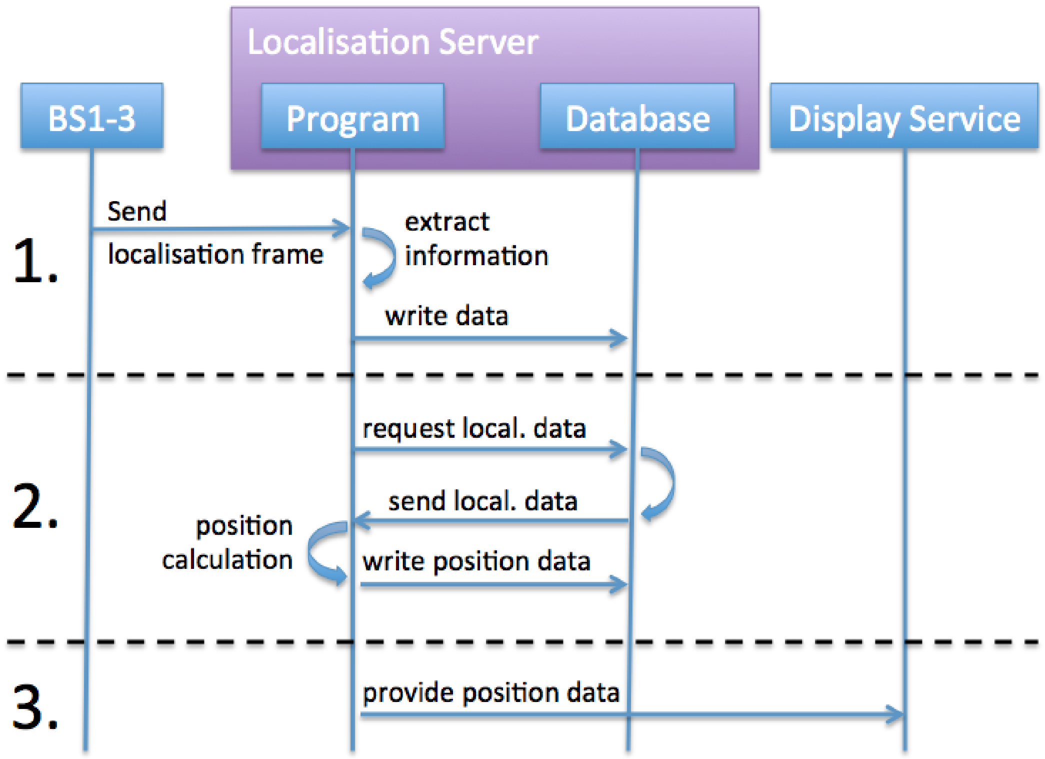


Figure 5.9: Server Sequence Diagram

Chapter 6

Localisation System Development

This chapter proposes an implementation of a localisation system based on the blueprint elaborated during the previous chapter. Due to time constraints and the absence of the hardware needed to build a mobile client and the base stations both devices have been realised as a piece of software. This enables the simulation of the developed localisation server software.

The general system processing is equivalent to the one depicted by the sequence diagram, figure 5.3 in chapter 5.1. Same holds true for the components and their interconnection. The client would be wirelessly connected to the base stations, which are connected to the local area network (LAN) via cable. The localisation server is also cabled to the LAN and the display service could run on the server or any other computer in the network. In the following sections the four main parts, base station, mobile client, localisation server and display service are treated separately. Because it is intended to use a modified version of a base station already developed by the OEAW (Austrian Academy of Science) for a future localisation system, their concept as well as the necessary adaptations are introduced. As already described, by trying to develop the mobile client, it turned out that the available hardware does not meet the requirements regarding the needed accuracy. Hence, the mobile client chapter contains a problem analysis discussing the problems that arise when trying to use a commercial off the shelf (COTS) hardware instead of developing a board from the scratch.

6.1 The Base Station (BS)

The blueprint chapter 5.1.1 describes the requirements for a base station that can be used for localisation purposes. Additionally three important extensions in comparison to a normal base station are described: Firstly, a scanner on the media independent interface (MII) that scans the incoming frames for localisation ones. Secondly, the a time-stamping functionality that allows the system to add timestamps to the outgoing and incoming frames. And thirdly, separated send and receiving paths with own antennas to avoid the need of switching the antenna when changing from sending to receiving mode or vice versa.

These general requirements are fulfilled by the base station device called SMiLE receiver developed at the Austrian Academy of Sciences (OEAW) during the $^{\text{flex}}$WARE project. These are aiming a TDoA (Time Difference of Arrival) localisation that requires clock synchronization between the base stations and the mobile client. In addition [4] introduces the use of the DTDoA (Differential Time Difference of Arrival) method which needs no clock synchronisation. But in every case, for clock synchronization as well for TDoA or DTDoA a filtering of the received packets as well as a time-stamping exact to the nanosecond is required. Therefore the proposal is to reprogram the already existing especially for high precision localisation developed hardware instead of developing an own one up from the ground.

6.2 The Mobile Client (MC)

When it came to develop a mobile client it relatively fast turned out that the available commercial of the shelf (COTS) hardware was not sufficient to even nearly reach the needed accuracy. Therefore it has been decided to concentrate on the development of a localisation server and to realize the hardware as a piece of software to simulate the server. Hence, the first part of this section is an analysis of the problems by trying to use the COTS hardware and the second describes the developed piece of software that simulates the client as well as the base stations by simply sending localisation frames, definable by the user, to the server.

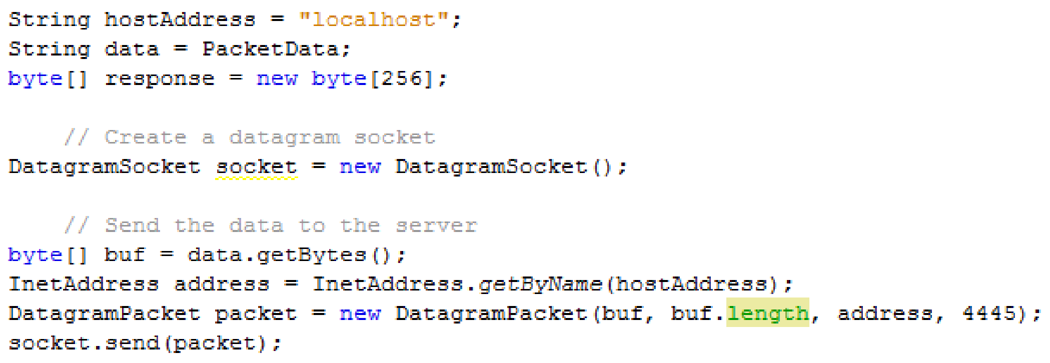
6.2.1 Problem Analysis

The problem analysis is based on a discussion about the possibilities to use a Microchip 16-Bit controller (product no.: dsPIC33FJXXXGPX06/X08/X10) in combination with a Microchip RF transceiver module (product no.: MRF24 WB0MA/MRF24WB0MB) for the planned localisation system. First weakness of the controller is the relatively low clock speed of the primary oscillator which has a maximum of 40 MHz and therefore is not sufficient for exact to the nanosecond measurements. The programming languages which can be used to write software for the controller are C and Assembler. Both could by used since C has no automatic memory management, which is one of the main problems for real-time applications. But since the result of the compilation from C to machine code is not known exactly, the execution time for the frame processing has to be measured, for example via an oscilloscope. But this entails an oscilloscope having a resolution that is high enough to measure with the required accuracy, which was not available. Another problem, that is located at the transceiver is that it is equipped with only one antenna. Now, when trying to send the package the device is first listening to the medium if it is free. If this is the case, the antenna has to be switched from the receiving to the sending mode. The time to perform this change between receiving and transmitting is denoted with 10 microseconds. A more precise specification in nanoseconds or a separated send path with an own antenna would be preferred in order to better calculate the overall processing time.

All these problems, which has already been figured out through the first discussion as well as the time constraints for this work let to the decision to concentrate on the server software and to implement a client software. This is now replacing the base station as well as the mobile client and is used for validating the server software functionalities.

6.2.2 Software Implementation

Because neither a working base station nor a mobile client were available a piece of software has been written to simulate the communication with the localisation server. This piece of software simply opens a network connection and sends localisation frames to the server. The timestamps, which define the measured distance from the client to the base stations are entered by the user. This allows to simulate various localisations. The network communication has been realised by using the UDP (User Datagram Protocol) protocol. This is a simple and connectionless protocol without message acknowledgement. Furthermore, the messages are not ordered. But since the data sent by the base stations will not exceed the size of one packet and a message received acknowledgement can be sent manually, these disadvantages aren't an issue. The Java code for establishing a socket and sending a packet is depicted in figure 6.1.



```
String hostAddress = "localhost";
String data = PacketData;
byte[] response = new byte[256];

    // Create a datagram socket
DatagramSocket socket = new DatagramSocket();

    // Send the data to the server
byte[] buf = data.getBytes();
InetAddress address = InetAddress.getByName(hostAddress);
DatagramPacket packet = new DatagramPacket(buf, buf.length, address, 4445);
socket.send(packet);
```

Figure 6.1: Establishing a Java UDP socket connection

The format of the data sent in the frame is as followed:
#loc#*locID*#*bsID*#*mcID*#*Timestamp 1*#*Timestamp 2*##

The first hash marks the start of the data, followed by "loc" to indicate the packet as a localisation packet. After another hash follows the localisation ID which identifies the localisation of one mobile client. Hence, a server has to receive three packets with the same localisation ID to be able to start the localisation process. The "locID" it the first variable data. From now on every data value is separated by a hash. Next are the base station Id (bsID), the mobile client ID (mcID), the first timestamp, which has been recorded as soon as the packet left the base station and the second timestamp, which has been recorded when the packet arrived the base station, after being sent by the mobile client. The end of the data is marked with two hashes. In this case, where no base station exists the data can be entered into the program to simulate different scenarios.

6.3 The Localisation Server

The server sequence diagram 5.9 of chapter 5.1.3 was the basis for the server software development. The localisation server firstly consists of a Java program listening for localisation frames, exchanging information with the database and calculating the mobile clients positions. Secondly it consists of the database that stores the raw data from the frames, the calculated mobile clients positions as well as several information about the localisation environment. The graphical user interface (GUI) that is realised as an independent piece of software and could run at any computer in the network is introduced in section 6.4. Both, the server software as well as the graphical user interface have been realised in Java. The decision to use Java was mainly based on the easy implementation of network communications, the possibility of creating multithreaded programs and on the flexibility to use Java with several operating systems. The last point was important because the future operating environment and therefore also the system to run the software are unknown. The use of several threads is interesting because of two points: first, to be able to benefit from a multi core processing power of the todays computing systems (although if the program is not resource intensive) and second, for being able to parallelize the different tasks, for example to be able to receive packets while writing their information to the database. And a network socket implementation is needed to listen for localisation packets from the base stations or in this case from the client software.

6.3.1 Server Software

The Server Software contains the following parts (see also figure 6.2):

- The Server class containing the main() and starting the server thread and the graphical user interface.

- The Server thread, listening for localisation packets, starting the thread for writing the frame data to the database and starting the thread for calculating the mobile clients position.

- The write raw data thread for extracting the needed information from the frame and writing them to the database.

- The localisation thread which is started after the raw data of all three base stations has been written and calculates the mobile clients position.

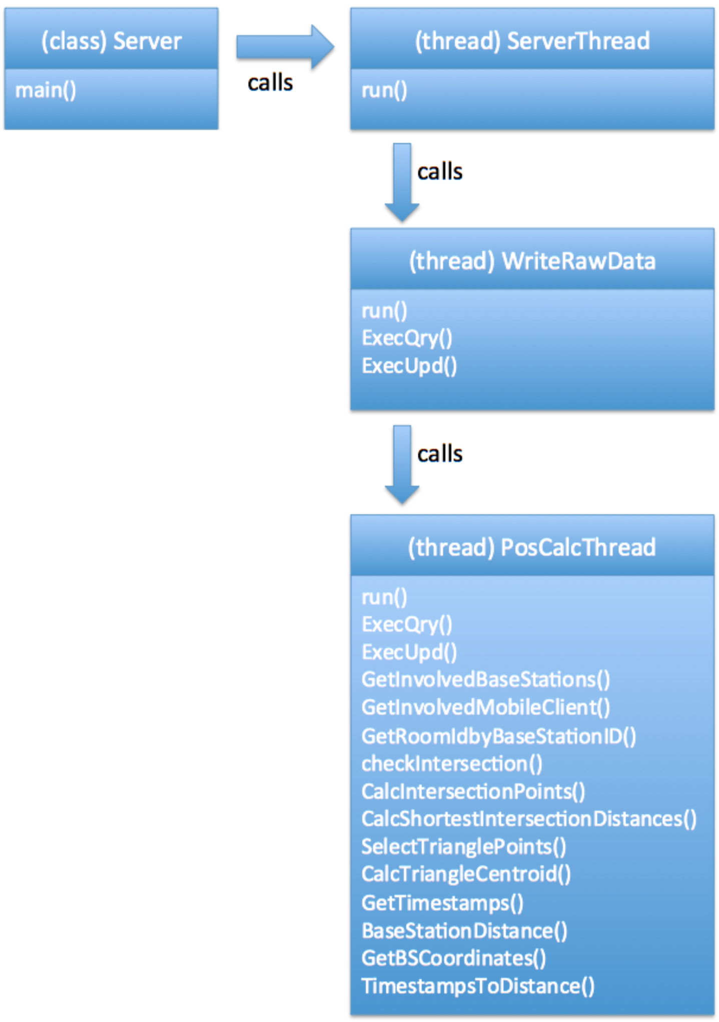
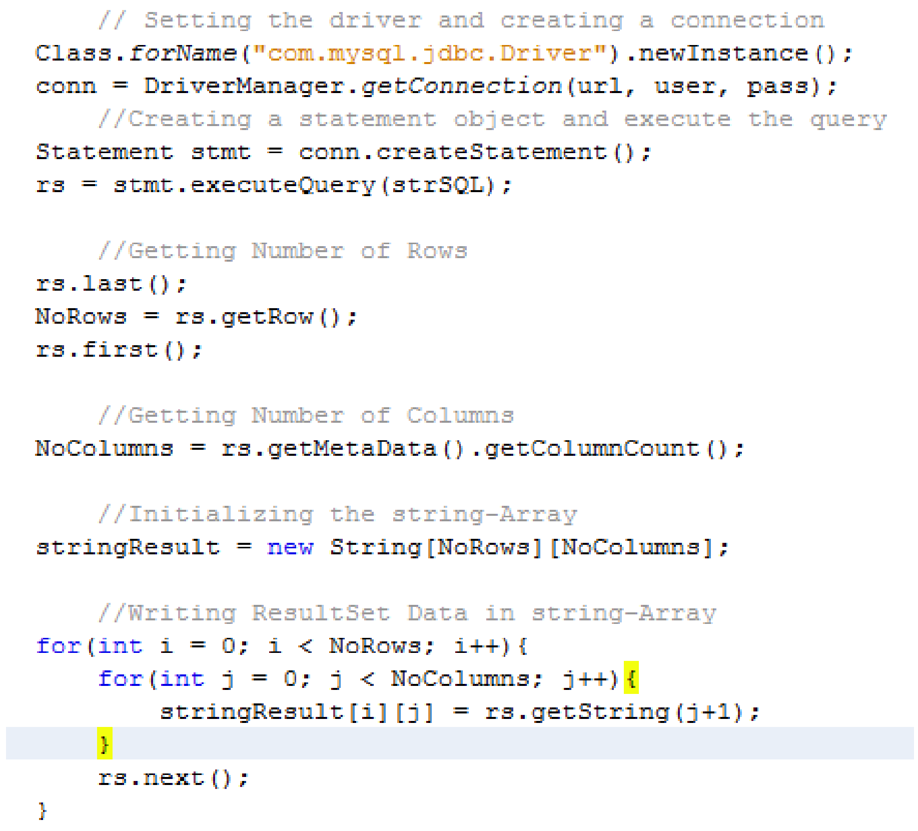


Figure 6.2: The Server Classes

The server program implements a UDP socket similar to the one introduced in 6.2.2 to be able to receive the localisation frames. As soon as a frame has been received, the server answers with a message received acknowledgement to the base station. Afterwards it starts the thread to write the information contained in the localisation frame to the database. This is done by firstly splitting the string in its single values. Secondly, the program checks if already three localisation frames for the certain localisation ID have been received and if the combination of the localisation ID and the base station is already stored in the database. This avoids wrong database entries for example if a base station accidentally has sent its data twice. If the checks have been successful the data are written. When executing a query or an update on the database several steps have to be taken. To avoid repeating the code for opening a database connection, executing the query and receiving the data again and again this whole part has been outsourced to own functions. One for the execution of a query (ExecQry) and one for the execution of an update (ExecUpd). The *ExecQry* function simply takes the SQL[1] string and delivers the answer from the database as a two dimensional string

[1]Subscriber Query Language

array representing the SQL result set. The *ExecUpd* function has been realised in similar way, merely that no values are returned: it only contains receiving the SQL string, opening a database connection, executing the update, committing it and closing the connection again. An example for how to query a database in Java is depicted in 6.3. The execution of an update is illustrated in 6.4. Further information about the database (e.g. the database-model) can be found in section 6.3.2. After the raw data is written the thread is finished and the program returns to the server thread.

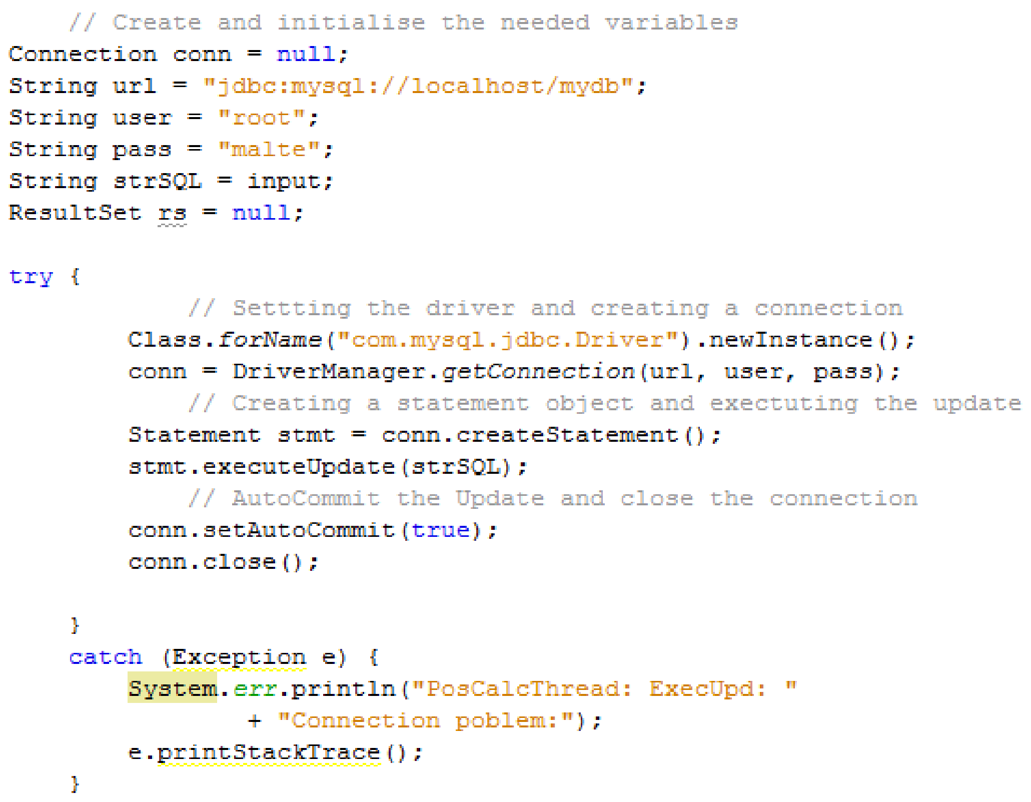


```
    // Setting the driver and creating a connection
Class.forName("com.mysql.jdbc.Driver").newInstance();
conn = DriverManager.getConnection(url, user, pass);
    //Creating a statement object and execute the query
Statement stmt = conn.createStatement();
rs = stmt.executeQuery(strSQL);

    //Getting Number of Rows
rs.last();
NoRows = rs.getRow();
rs.first();

    //Getting Number of Columns
NoColumns = rs.getMetaData().getColumnCount();

    //Initializing the string-Array
stringResult = new String[NoRows][NoColumns];

    //Writing ResultSet Data in string-Array
for(int i = 0; i < NoRows; i++){
    for(int j = 0; j < NoColumns; j++){
        stringResult[i][j] = rs.getString(j+1);
    }
    rs.next();
}
```

Figure 6.3: Querying a Database with Java

After the frames from all three base stations have been received and written to the database the position calculation thread starts. The calculation is a geometric trilateration based on [20], which is described in the next section in detail. The calling server thread passes the localisation ID to the position calculation thread. Then, the first step is to query the mobile client, the base stations and the room involved in the localisation. The room ID is derived by looking at the data base table*tblBaseStation* that contains all base station related information, inherent the ID of the room the respective base station is installed in.

Next step is to check if the radius around the base stations, emerging by the measured distance from the base stations to the client, are intersecting. In the theoretical case where all measure are infinitely precise, all circles would intersect in exactly one point, where the mobile client is located. Based on the measurements inaccuracy this will not be the case and instead of delivering an exact point for the desired position the intersections deliver an



```java
    // Create and initialise the needed variables
Connection conn = null;
String url = "jdbc:mysql://localhost/mydb";
String user = "root";
String pass = "malte";
String strSQL = input;
ResultSet rs = null;

try {
            // Settting the driver and creating a connection
        Class.forName("com.mysql.jdbc.Driver").newInstance();
        conn = DriverManager.getConnection(url, user, pass);
            // Creating a statement object and exectuting the update
        Statement stmt = conn.createStatement();
        stmt.executeUpdate(strSQL);
            // AutoCommit the Update and close the connection
        conn.setAutoCommit(true);
        conn.close();

    }
    catch (Exception e) {
        System.err.println("PosCalcThread: ExecUpd: "
                + "Connection poblem:");
        e.printStackTrace();
    }
```

Figure 6.4: Execute an update on a Database with Java

area in which the client is located in. The size of this area depends on the inaccuracy which shouldn't be more than a two meter radius, as described in the requirements in chapter 2. If two circles are not intersecting at all any measurement error has occurred and the position calculation can not start immediately. In this case the measurement could be repeated or the calculation has to be functionally extended as described in the algorithm extension section.

Now, the six intersection points of the circles have to be calculated. Then the distances among all points are measured and sorted by their length to obtain the three shortest distances which are shaping a triangle. This triangle formed area is where the mobile client is located in. For deriving an exact point, the centroid of the triangle is calculated and assumed as its position.

The Trilateration Algorithm

The following section contains the description of the mathematical part as well as the algorithmic Java implementation. The mathematical part is based on [20]. The code of the program can be found in appendix C.

Figure 6.5 depicts an example case to point out the needed calculations, where the six needed intersections points are marked with red circles. 6.6 illustrates two of the circles as well as the points and lines that have to be calculate.

The one circles center is BS1 = ($BS1_x$,$BS1_y$) with the radius $BS1_r$. The other circles center is BS2 = ($BS2_x$,$BS2_y$) with the radius $BS2_r$. First target is to find the circles intersection

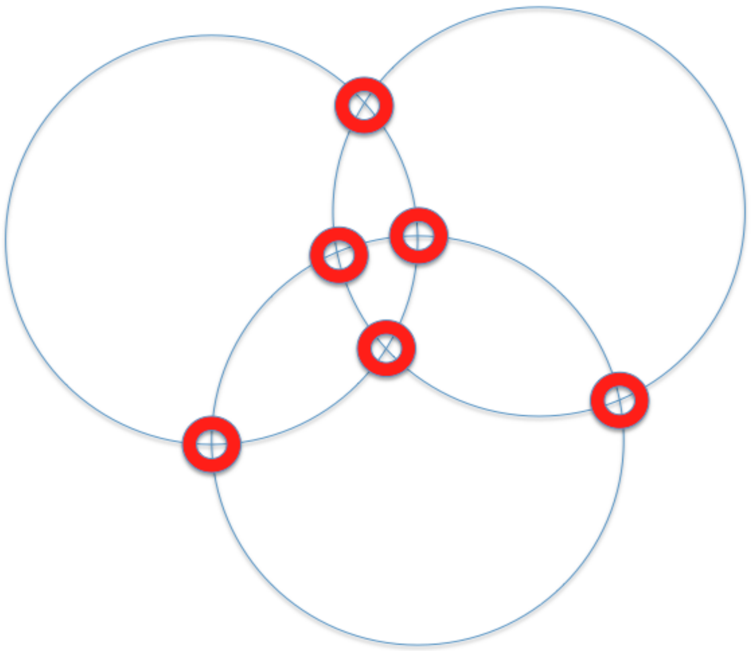


Figure 6.5: Example case with marked intersection points

points A and C. Therefore the distance between the circles centers (Δ) has to be calculated first:

$$\Delta_x = BS1_x - BS2_x \tag{6.1}$$

$$\Delta_y = BS1_y - BS2_y \tag{6.2}$$

$$\Delta^2 = \Delta_x^2 + \Delta_y^2 \tag{6.3}$$

$$\Delta = \sqrt{\Delta_x^2 + \Delta_y^2} \tag{6.4}$$

If $\Delta > BS1_r + BS2_r$ then both circles do not intersect. Next step is to find point B = (b_x,b_y), which is located at the intersection of the line between $BS1_r$ and $BS2_r$ as well as the line between A and C:

$$s^2 + u^2 = BS1_r^2 \tag{6.5}$$

$$t^2 + u^2 = BS2_r^2 \tag{6.6}$$

with: s = Distance between BS1 and B
t = Distance between BS2 and B
u = Distance between A and B or B and C

By subtracting both equations:

$$s^2 + u^2 - t^2 - u^2 = BS1_r^2 - BS2_r^2 \tag{6.7}$$

$$s^2 - t^2 = BS1_r^2 - BS2_r^2 \tag{6.8}$$

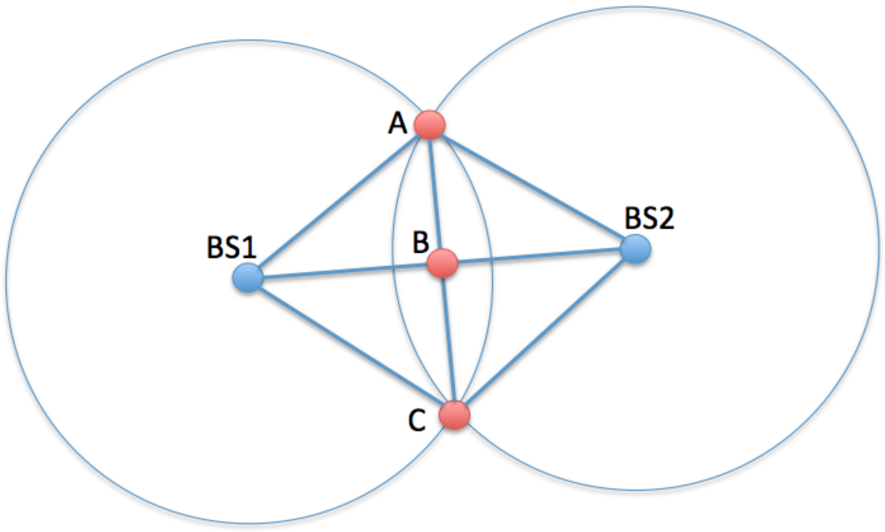


Figure 6.6: Two intersecting circles

Factor the left side:

$$(s-t)(s+t) = BS1_r^2 - BS2_r^2 \tag{6.9}$$

Because $s + t = \Delta$ and therefore $t = \Delta - s$, equation 6.9 can be written like:

$$(s-(\Delta - s))\Delta = BS1_r^2 - BS2_r^2 \tag{6.10}$$
$$(s-\Delta + s)\Delta = BS1_r^2 - BS2_r^2 \tag{6.11}$$
$$2\Delta s - \Delta^2 = BS1_r^2 - BS2_r^2 \tag{6.12}$$
$$s = \frac{\Delta^2 + BS1_r^2 - BS2_r^2}{2\Delta} \tag{6.13}$$

By knowing s, we can solve following equations to derive the coordinates of point B:

$$b_x = BS1_x + \frac{\Delta_x s}{\Delta} \tag{6.14}$$
$$b_y = BS1_y + \frac{\Delta_y s}{\Delta} \tag{6.15}$$

By having equation 6.13 we can change equation 6.5 and are able to solve it:

$$u = \sqrt{BS1_r^2 - s^2} \tag{6.16}$$

Then the coordinates of point A can be described as:

$$a_x = b_x - \frac{\Delta_y u}{\Delta} \tag{6.17}$$
$$a_y = b_y + \frac{\Delta_x u}{\Delta} \tag{6.18}$$

Furthermore, the coordinates of point C are:

$$c_x = b_x + \frac{\Delta_y u}{\Delta} \tag{6.19}$$

$$c_y = b_y - \frac{\Delta_x u}{\Delta} \tag{6.20}$$

This was the part which is described in [20] and which is implemented in some of the functions of the position calculation thread depicted in 6.7, 6.8, 6.9 and 6.10. Those four pictures illustrate the order of calling the different functions while the position calculation thread is running. The whole process depicted in 6.9 is executed three times, once for the circles of base station one and two, once for base station one and three and once for base station two and three. At the diagrams where the run() function starts several different actions these are marked in different colors (blue and dark) for a better readability.

By processing the calculation for all three circles as just described, the six intersection points of figure 6.5 are derived. Now, the three points in the middle, forming the triangle in which the client is located, have to be found. Therefore, the x-y-coordinates of all 6 intersection points are handed to the *CalcShortestIntersectionPoints()* function. Here, the fifteen possible lines between the points are calculated and written in a two dimensional matrix of the following form: One row represents one line and contains five columns for the x-coordinate of the start point, the y-coordinate of the start point, the x-coordinate of the end point, the y-coordinate of the end point and the distance between both of them. Afterwards a one dimensional reference array containing the numbers from zero to fourteen (one for every row of the before described matrix) is created. Now this reference array is sorted by referencing to the distances in column five of the two-dimensional array. Then the three shortest lines and their x-y-coordinates are written into a matrix which is handed back to the calling function as return value. The *SelectTriangleCoordinates()* function takes those six coordinates (six x- and six y-coordinates) and eliminates the equal ones so that only the three triangle coordinates are left. Based on those the *CalcTrianglePoints()* function calculates the centroid of the triangle which is assumed as the position of the mobile client.

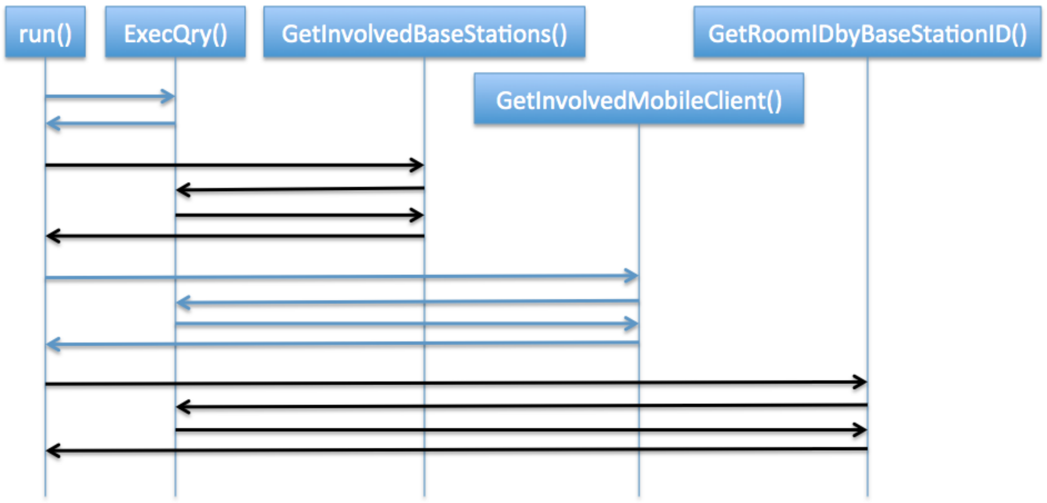

run()
ExecQry()
GetInvolvedBaseStations()
GetRoomIDbyBaseStationID()
GetInvolvedMobileClient()

Figure 6.7: Position calculation: sequence diagram part 1 of 4

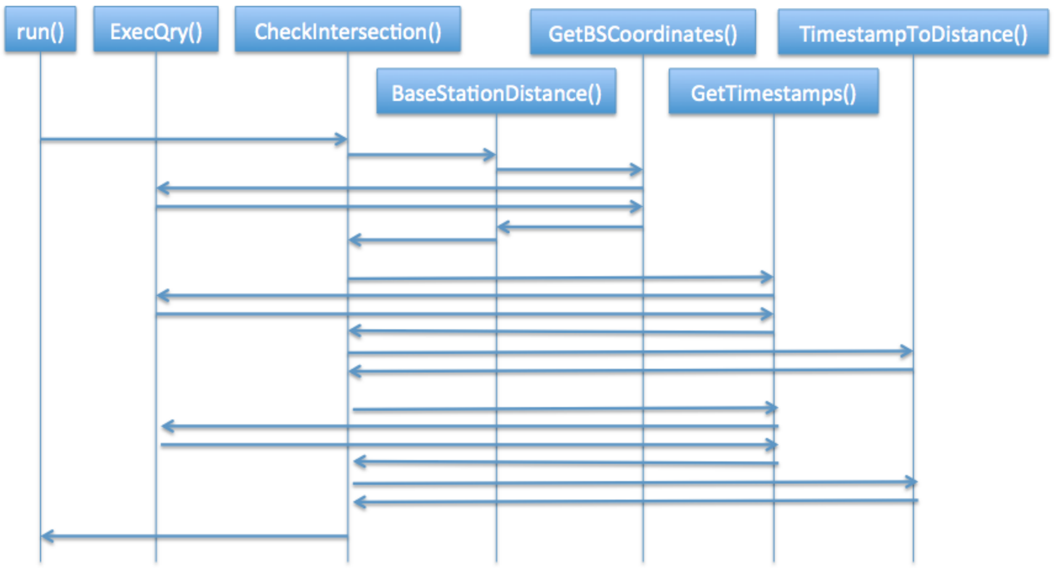


Figure 6.8: Position calculation: sequence diagram part 2 of 4

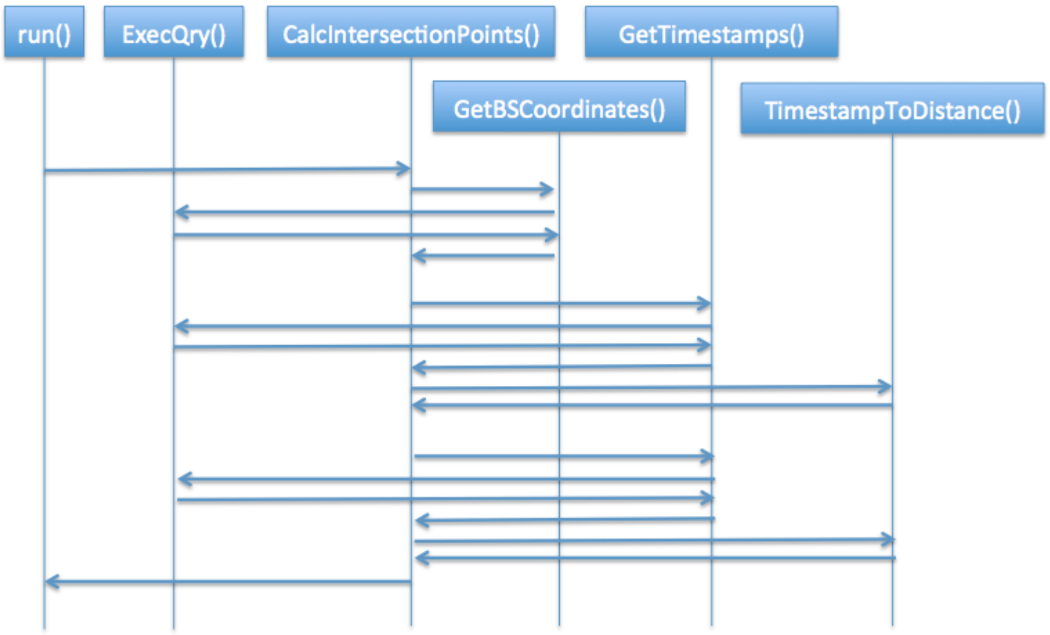


Figure 6.9: Position calculation: sequence diagram part 3 of 4

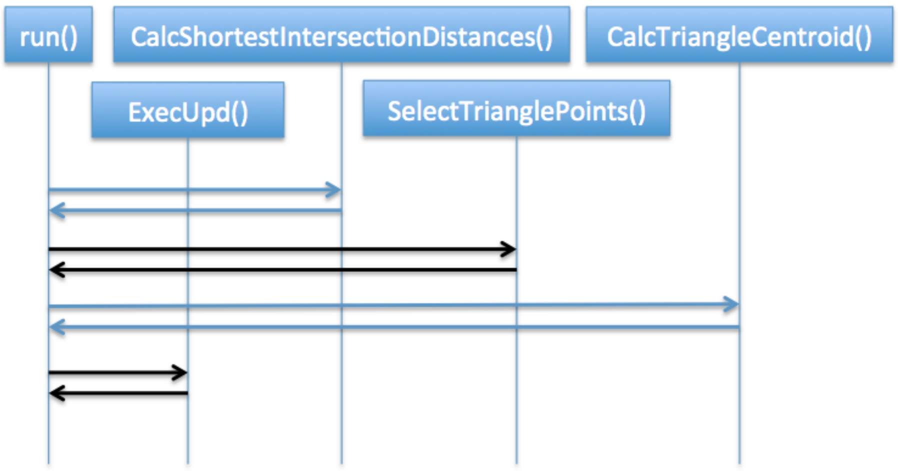


Figure 6.10: Position calculation: sequence diagram part 4 of 4

Future Algorithm Extensions

The before introduced localisation algorithm is just a basis which should be enhanced for use in a real case scenario. In the following some important extensions are proposed:

Enable a localisation even if two circles around the base stations, resulting from the distance of the client to the base station, are not intersecting each other. Now, in the beginning of the localisation process, a function is checking if all three circles are intersecting each other. If this is not the case, the localisation has to be cancelled. To avoid this, there are at least two possibilities:

1. Simply restart the whole localisation to proof the measures.

2. Adapt the algorithm to deal with the situation. This includes first to check if a localisation within the required accuracy is still possible. Secondly, calculating the position based on another reference point as for example shown in figure 6.11: Calculating point y which is on the circle around base station three and at the same time on the line between the center of the base station and the intersection point x of the two other base stations. Then the position of the client could be referenced to as the middle of the the line between x and y.

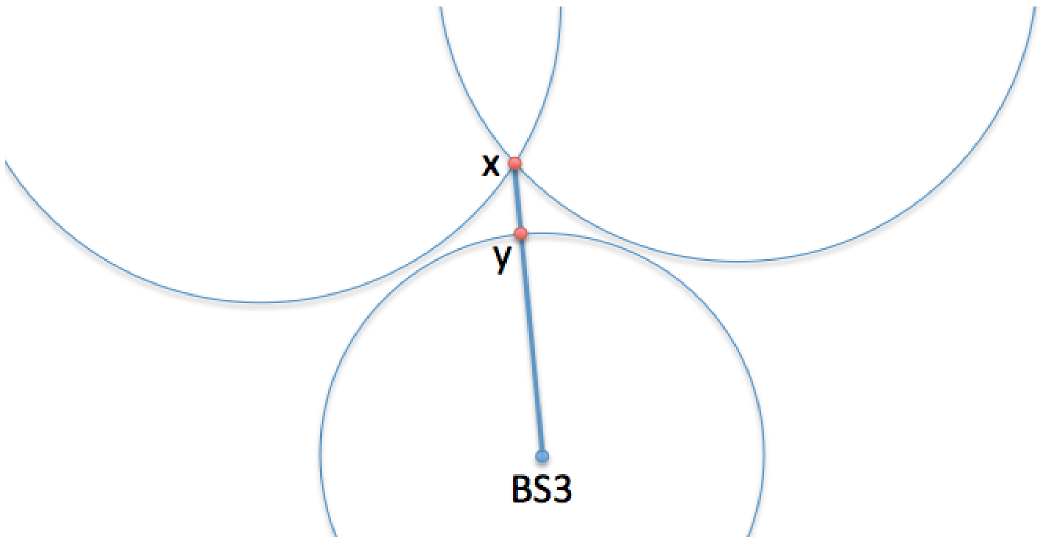


Figure 6.11: Non intersecting circles

Another extension could be to increase the localisation accuracy by enabling the use of more than three base stations. In the first step by using the ones in the room, if existing and in the second step by trying to use base stations from other rooms, too. Second case is definitely very challenging because of the signal change when it has to pass a wall. However, this case is also very important to avoid equipping every small room with three base stations. So there have to be a prioritization that favors the base stations from within the room where the client is located when doing the calculation and give a higher weight to theirs measurements.

Furthermore, it needs a better accuracy evaluation that measures if the area, resulting from the intersecting circles, where the client is suspected isn't larger than the two meter radius which is set as a requirement. Moreover, the size of the area of uncertainty could be presented to the user so that he / she is not loosing sight of the fact that there is a certain unprecisenes regarding the patients position. Because if only a dot on map is presented, one could easily forget that most certainly the client is located within a specific area and not exactly at the presented point.

6.3.2 The Database

A MySQL database is used to store the localization information. Figure 6.12 depicts the created tables and their connections.

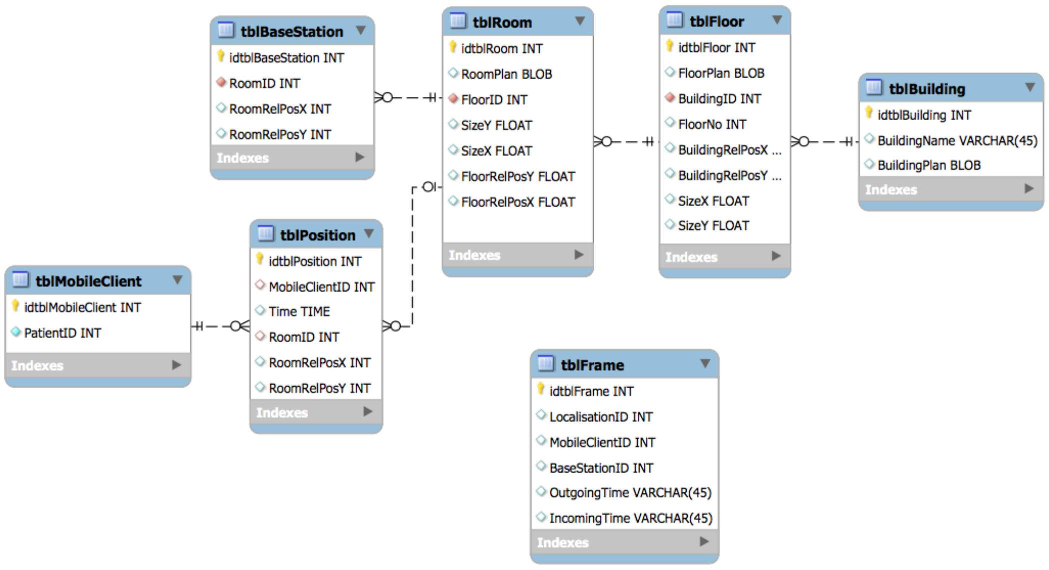


Figure 6.12: Database Model

The database is logically split into two parts. First contains only the table *tblFrame* which stores the raw data from the localisation frames. There are fields for recording the localisation ID, the mobile client ID, the base station ID and the outgoing and incoming timestamp. Three base stations each sending one localisation frame are needed for the localisation of one mobile client. Hence, the table has to contain three entries for one clients localisation. These entries will have the same localisation ID and obviously the same mobile client ID.

The second part of the database contains the rest of the tables. Four of them, the *tblBuilding*, the *tblFloor*, the *tblRoom* and the *tblBaseStation*, are representing the infrastructure of the localisation area. That could be a hospital building containing different floors, with several rooms and each room equipped with three base stations. The first of the two tables left is the *tblMobileClient* and represents the existing mobile clients and enables an association to the patient using it. The other one, the *tblPosition* stores the information calculated by the positioning algorithm. This is the position of a specific mobile client at a specific time.

6.4 The Display Service

For providing a graphical user interface to the user, a piece of software has been programmed. Regarding the time constraints of this work, the user interface is a very simple piece of software that always shows the last localisation entry. It is completely independent from the other programs to be able to run on any machine in the network. It only needs access to the localisation database to read the information about the last localisation and the other needed information, like the involved base stations and their positions as well as the information about the room, from the database. Afterwards the coordinates are converted from the cartesian coordinate system to the system used in Java that starts in the upper left corner. When having the right coordinates for all components those are drawn by using the proper AWT/SWING functions. This leads to an output depicted in figure 6.13.

When the GUI gets started, now it only shows the last localisation saved to the database.

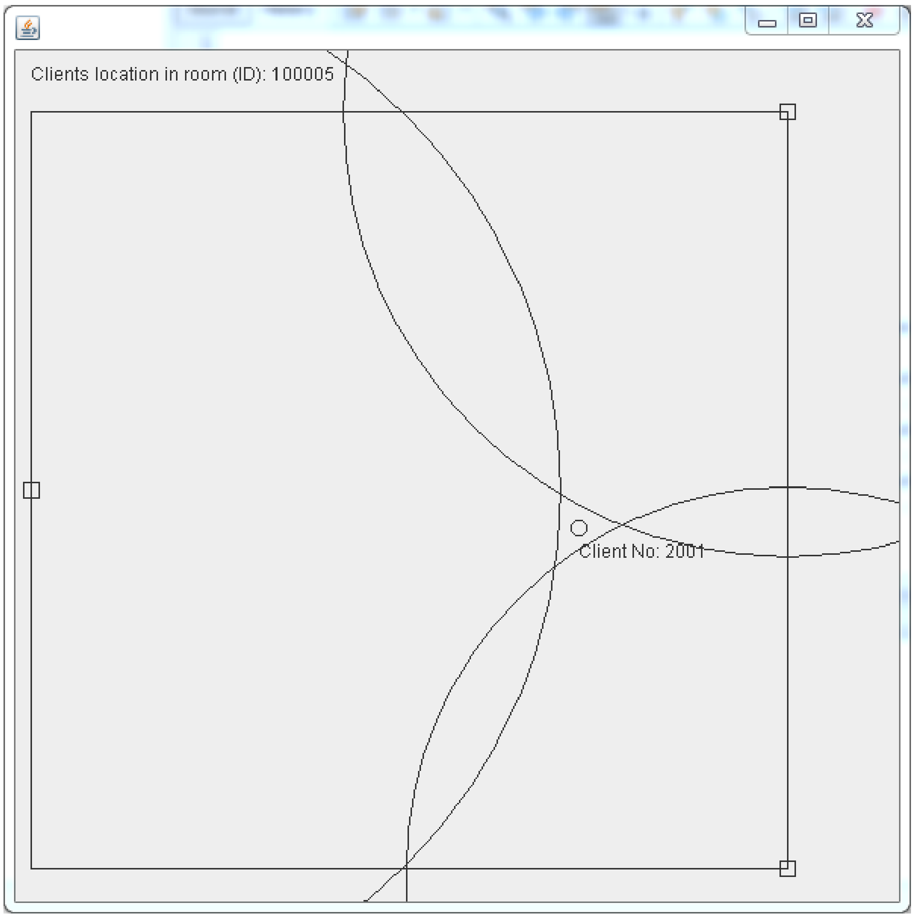


Figure 6.13: The Display Service

For improving the GUI[2] it should be able to interact with the user. This includes not only showing the latest localisation but also specific clients at specific times as well as at which floor of which building and where at this floor the showed room room is located. A first and very easy step for improving it could be a time triggered check for new localisation entries in the database.

A simple further development would be to implement the display service as a web interface. Because the software is already written in Java, the effort to change it to an applet is relatively low. But the benefit for not being forced to install the software on every device that should be able to show the patients location is immense, since only the server running the web service has to be serviced.

[2]Graphical User Interface

Chapter 7

Validation

The validation chapter proofs the correctness of the calculations executed by the developed algorithm. Therefore, four cases has been developed with a CAD[1] software. These show the defined room (see also appendix D) including the base stations and different measurements for each given case. The measurements are a specific distance which represents one measurement between the certain base station and the mobile client as well as all necessary distances. These distances was measured within the CAD software. For validating the algorithms performance the centroid coordinates, which are the result of the algorithm, are calculated by hand and compared with the algorithms results. In the following every section represents one case. Because the procedure is the same for every case, only the first one is described in detail.

7.1 Case One

Figure 7.1 depicts the first case created with the CAD software. The distances from the base stations are represented by an arc of a circle with the certain distance. Furthermore, the three intersection points are marked and measured regarding the x and y axis. Also the base stations positions can be read from the attached measurements. These measurements have been done within the CAD software. The size of the room as well the positions of the base stations correspond to the ones of the defined test case (see appendix D). These information are also stored in the database and are used when executing the developed software. The last result, which is based on all of the previous calculations, is the centroid of the triangle formed by the three intersection points. Therefore the centroid coordinates are calculated by hand. Afterwards, the program is executed with the distances defined in the CAD picture and the results are compared with each other.

7.1.1 Calculation of the Centroid coordinates by Hand

The calculation of a centroid is done by using the following formulas:

$$x_{centroid} = \frac{1}{3} * (x_{BS1} + x_{BS2} + x_{BS3}) \tag{7.1}$$

$$y_{centroid} = \frac{1}{3} * (y_{BS1} + y_{BS2} + y_{BS3}) \tag{7.2}$$

[1]Computer Aided Design

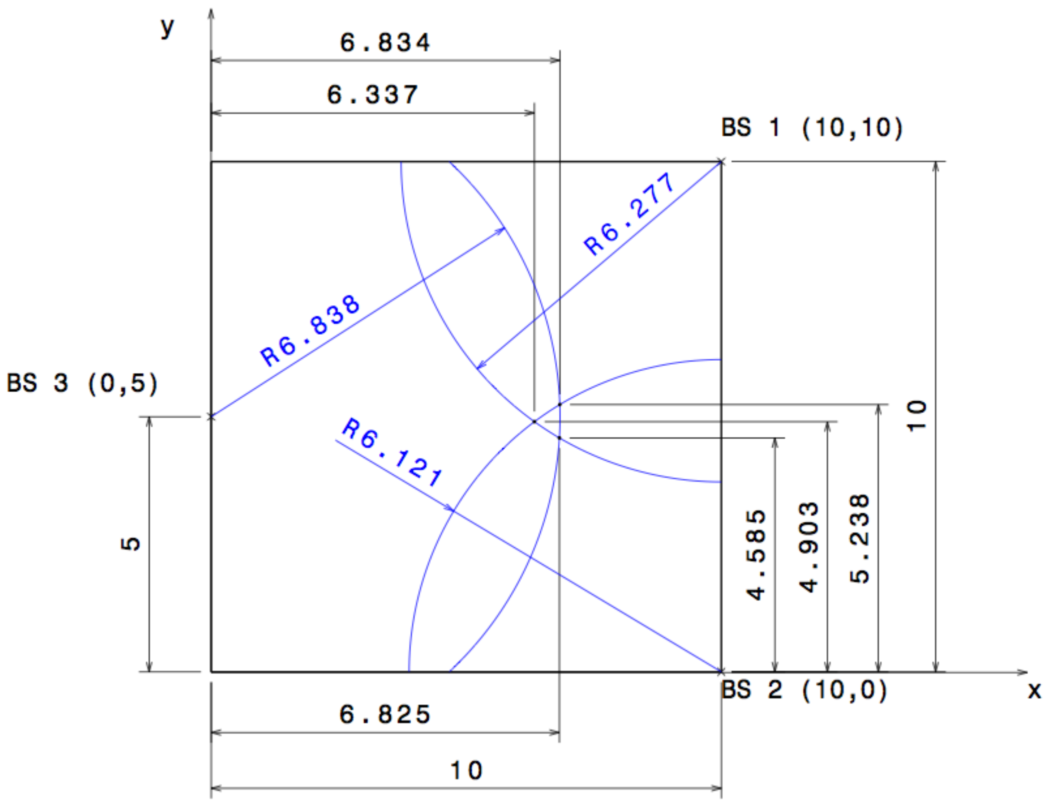
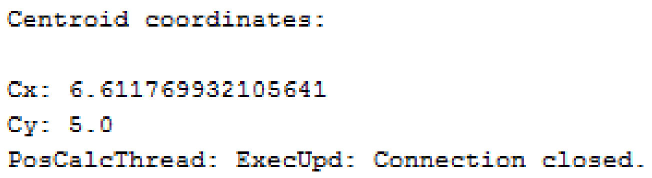


Figure 7.1: Case One: CAD Drawing



```
Centroid coordinates:

Cx: 6.611769932105641
Cy: 5.0
PosCalcThread: ExecUpd: Connection closed.
```

Figure 7.2: Case One: Centroid Coordinates

with:
x/y$_{\text{centroid}}$ = x/y-coordinate of the triangles centroid
x$_{\text{BS1-3}}$ = x-coordinate of base station one to base station three
y$_{\text{BS1-3}}$ = y-coordinate of base station one to base station three

Hence, the calculation looks as followed:

$$x_{centroid} = \frac{1}{3} * (6.834 + 6.337 + 6.825) = 6.665 \tag{7.3}$$

$$y_{centroid} = \frac{1}{3} * (5.238 + 4.903 + 4.585) = 4.908 \tag{7.4}$$

7.1.2 Execution of the Algorithm

When executing the client software that sends the timestamps to the localisation server, the user has to enter the measured distance between every base station and the mobile client in nanoseconds. Every nanosecond of elapsed time corresponds to 0.28cm travelled distance. Therefore, the radius defined at the CAD pictures have to be converted to nanoseconds. Because the client software only allows the input of full nanoseconds, which is assumed as the most accurate time to be able to measure, the converted values have to be rounded in most of the cases. Hence, the calculated results will vary a little bit from the ones calculated by hand.

$$BS1_{Radius} = 6.277m \mathrel{\widehat{=}} 22.41ns \Rightarrow 22ns \tag{7.5}$$

$$BS2_{Radius} = 6.121m \mathrel{\widehat{=}} 21.86ns \Rightarrow 22ns \tag{7.6}$$

$$BS3_{Radius} = 6.838m \mathrel{\widehat{=}} 24.42ns \Rightarrow 24ns \tag{7.7}$$

When entering those nanosecond values into the software following results are produced: Figure 7.2 depicts the centroid coordinates calculated by the position calculation algorithm. Figure 7.3 illustrates the output of the graphical user interface which can be compared to the CAD painting. By comparing both with the calculated data by hand or respectively with the CAD drawing, the results are the same up to a small error because of the rounding to full nanoseconds as done before.

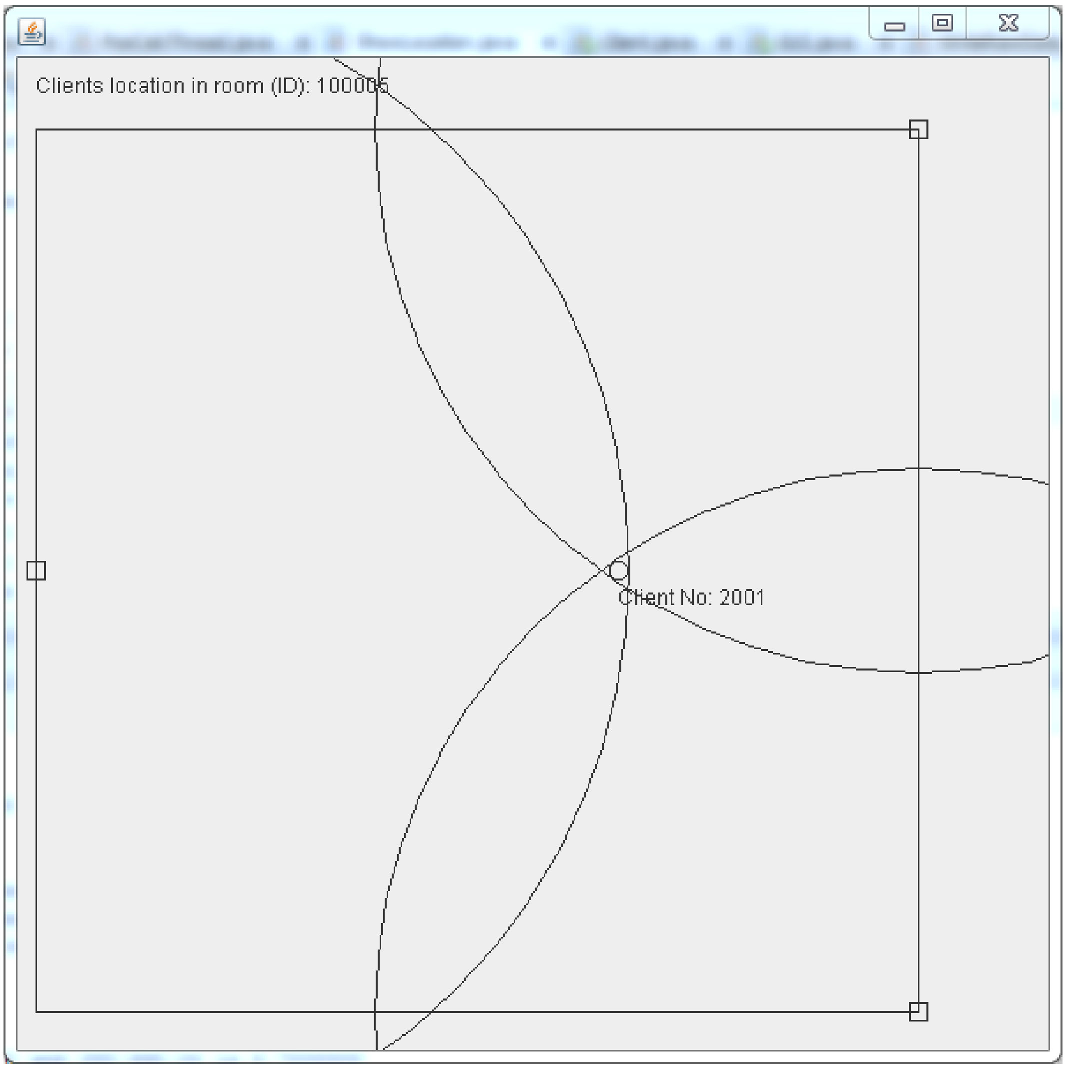


Figure 7.3: Case One: GUI Output

7.2 Case Two

Figure 7.4 depicts the actual case as a CAD drawing.

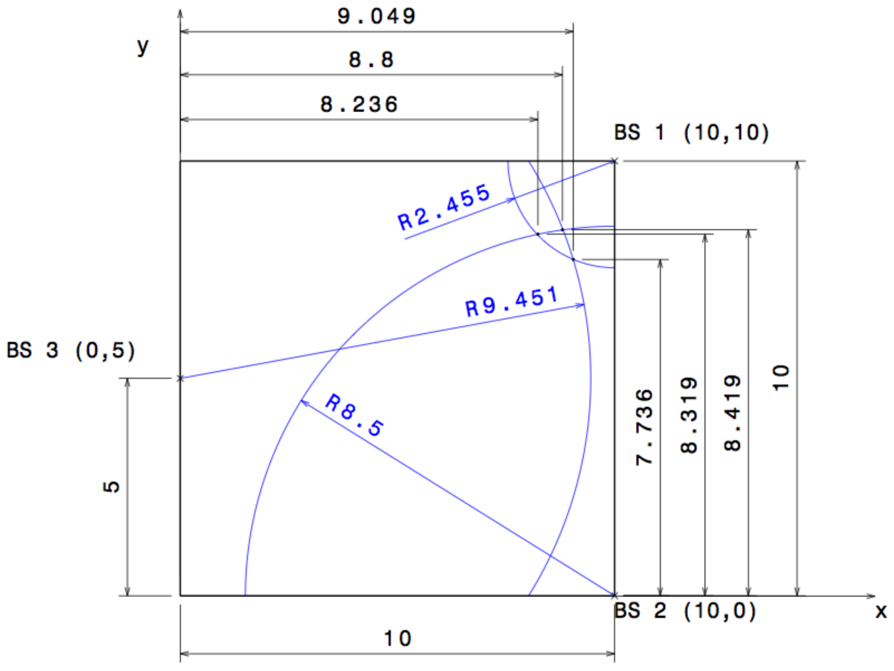


Figure 7.4: Case Two: CAD Drawing

7.2.1 Calculation of the Centroid coordinates by Hand

By taking the intersection coordinates of the CAD drawing, following x-y-coordinates of the centroid are calculated by hand:

$$x_{centroid} = \frac{1}{3} * (8.236 + 8.8 + 9.049) = 8.695 \tag{7.8}$$

$$y_{centroid} = \frac{1}{3} * (8.319 + 8.419 + 7.736) = 8.158 \tag{7.9}$$

7.2.2 Execution of the Algorithm

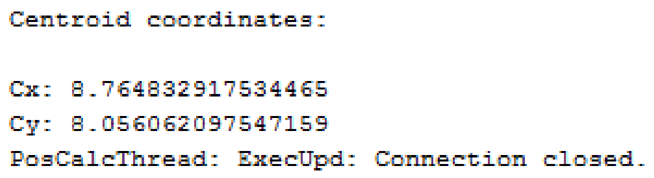
The distances of the base stations to the mobile client taken from the CAD drawing lead to following nanosecond values:

$$BS1_{Radius} = 2.455m \mathrel{\widehat{=}} 8.76ns \Rightarrow 9ns \tag{7.10}$$

$$BS2_{Radius} = 8.5m \mathrel{\widehat{=}} 30.35ns \Rightarrow 30ns \tag{7.11}$$

$$BS3_{Radius} = 9.451m \mathrel{\widehat{=}} 33.75ns \Rightarrow 34ns \tag{7.12}$$

The position calculation algorithm calculates the values depicted in 7.5. Figure 7.6 illustrates the output of the graphical user interface.



```
Centroid coordinates:

Cx: 8.764832917534465
Cy: 8.056062097547159
PosCalcThread: ExecUpd: Connection closed.
```

Figure 7.5: Case Two: Centroid Coordinates

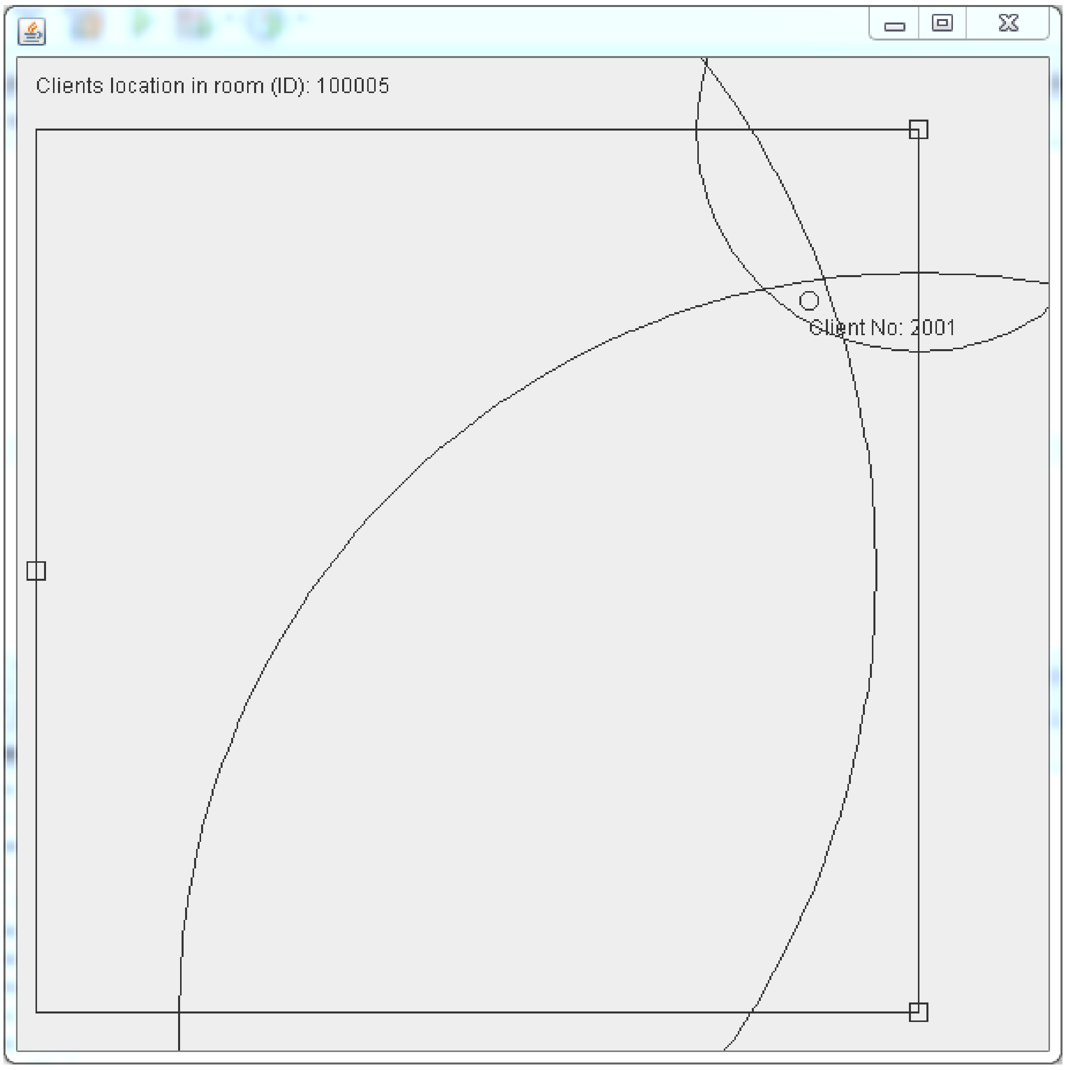


Figure 7.6: Case Two: GUI Output

7.3 Case Three

Figure 7.7 depicts the actual case as a CAD drawing.

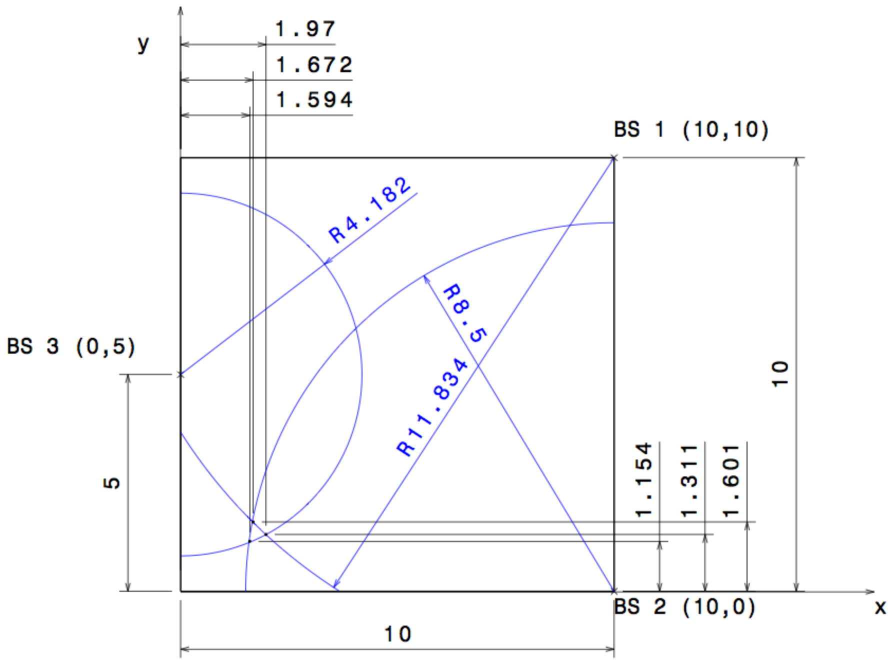


Figure 7.7: Case Three: CAD Drawing

7.3.1 Calculation of the Centroid coordinates by Hand

By taking the intersection coordinates of the CAD drawing, following x-y-coordinates of the centroid are calculated by hand:

$$x_{centroid} = \frac{1}{3} * (1.594 + 1.672 + 1.97) = 1.745 \quad (7.13)$$

$$y_{centroid} = \frac{1}{3} * (1.154 + 1.601 + 1.311) = 1.355 \quad (7.14)$$

7.3.2 Execution of the Algorithm

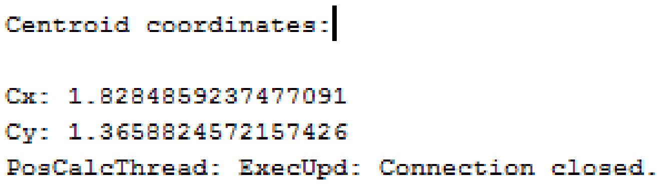
The distances of the base stations to the mobile client taken from the CAD drawing lead to following nanosecond values:

$$BS1_{Radius} = 11.834m \mathrel{\widehat{=}} 42.26ns \Rightarrow 42ns \quad (7.15)$$

$$BS2_{Radius} = 8.5m \mathrel{\widehat{=}} 30.35ns \Rightarrow 30ns \quad (7.16)$$

$$BS3_{Radius} = 4.182m \mathrel{\widehat{=}} 14.93ns \Rightarrow 15ns \quad (7.17)$$

The position calculation algorithm calculates the values depicted in 7.8. Figure 7.9 illustrates the output of the graphical user interface.



```
Centroid coordinates:

Cx: 1.8284859237477091
Cy: 1.3658824572157426
PosCalcThread: ExecUpd: Connection closed.
```

Figure 7.8: Case Three: Centroid Coordinates

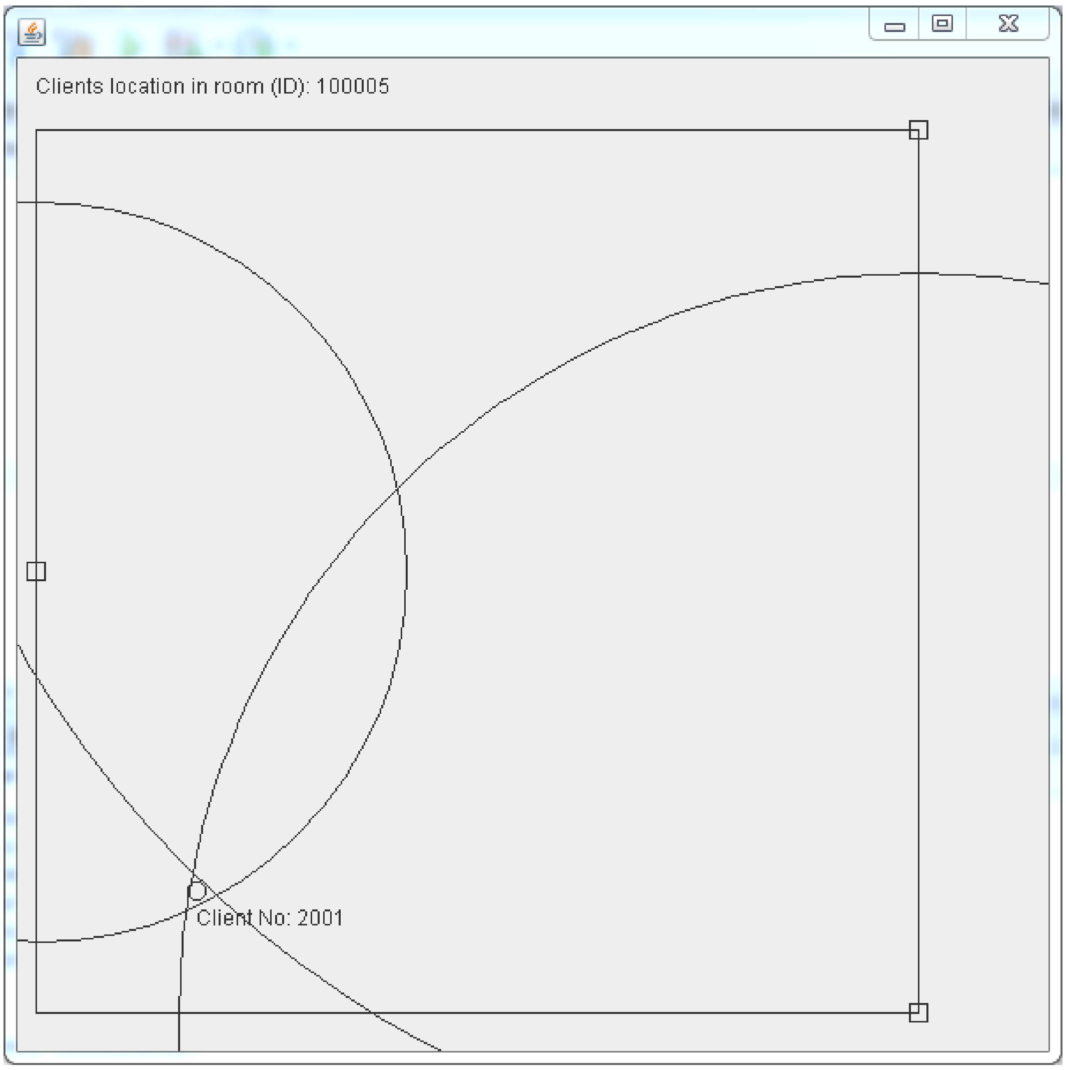


Figure 7.9: Case Three: GUI Output

7.4 Case Four

Figure 7.10 depicts the actual case as a CAD drawing. The singularity of this case in comparison to the ones before is that the three distances from the base stations are not overlapping in full. In this case these are near enough to form three intersection points but the localisation area is outside of the three circles as can be seen in the CAD painting. Nevertheless, the algorithm works in a proper way, because the centroid of the triangle, formed by the three intersection points, can be calculated.

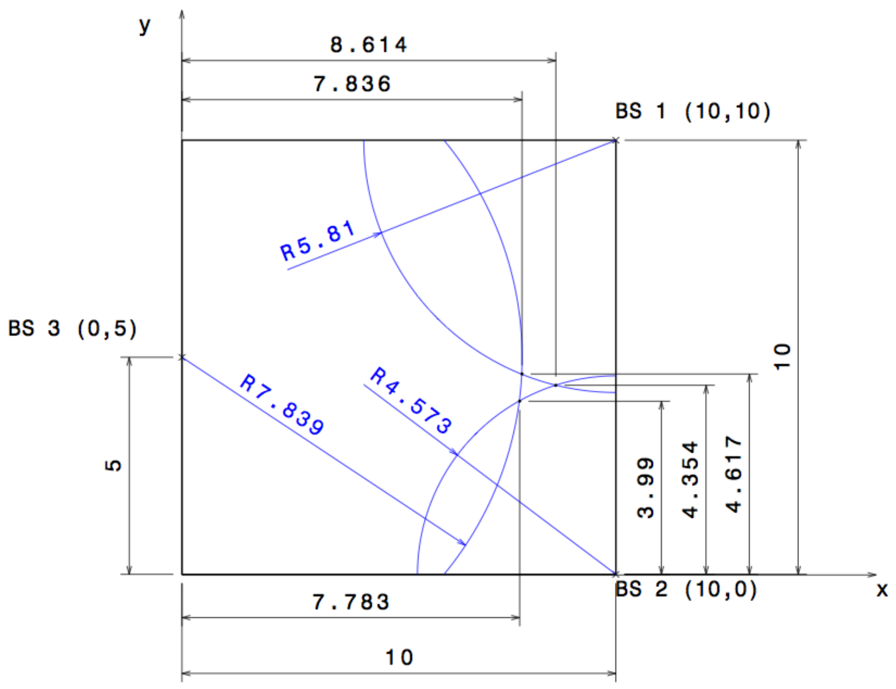


Figure 7.10: Case Four: CAD Drawing

7.4.1 Calculation of the Centroid coordinates by Hand

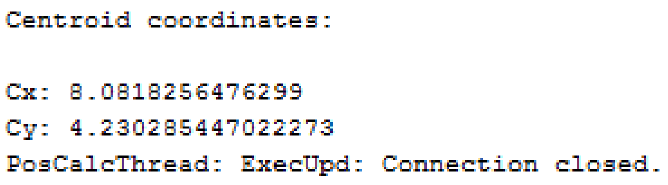
By taking the intersection coordinates of the CAD drawing, following x-y-coordinates of the centroid are calculated by hand:

$$x_{centroid} = \frac{1}{3} * (7.863 + 7.783 + 8.614) = 8.077 \tag{7.18}$$

$$y_{centroid} = \frac{1}{3} * (4.617 + 3.99 + 4.354) = 4.320 \tag{7.19}$$

7.4.2 Execution of the Algorithm

The distances of the base stations to the mobile client taken from the CAD drawing lead to following nanosecond values:



```
Centroid coordinates:

Cx: 8.0818256476299
Cy: 4.230285447022273
PosCalcThread: ExecUpd: Connection closed.
```

Figure 7.11: Case Four: Centroid Coordinates

$$BS1_{Radius} = 5.81m \mathrel{\widehat{=}} 20.75ns \Rightarrow 21ns \tag{7.20}$$

$$BS2_{Radius} = 4.573m \mathrel{\widehat{=}} 16.33ns \Rightarrow 16ns \tag{7.21}$$

$$BS3_{Radius} = 7.839m \mathrel{\widehat{=}} 27.99ns \Rightarrow 28ns \tag{7.22}$$

The position calculation algorithm calculates the values depicted in 7.11. Figure 7.12 illustrates the output of the graphical user interface.

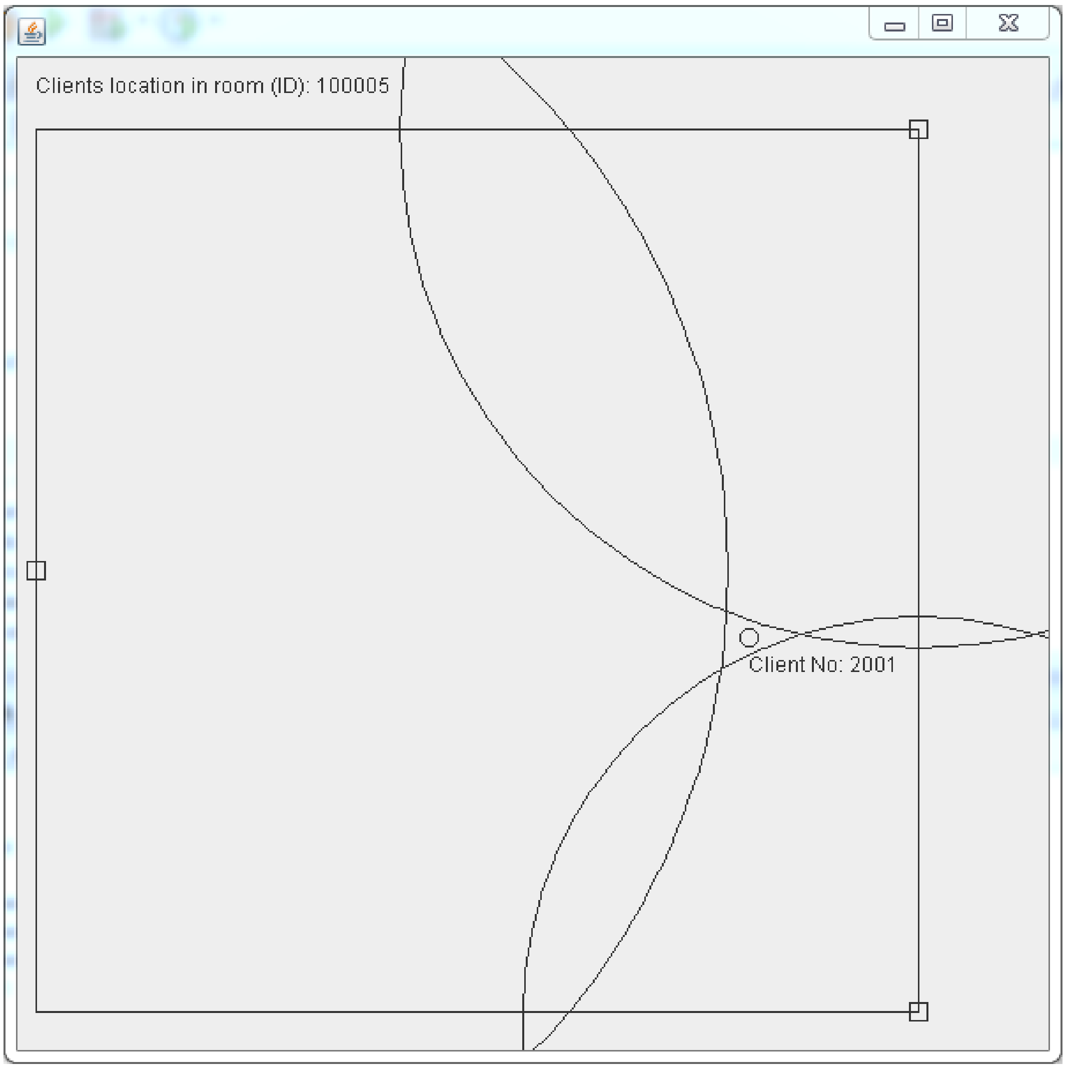


Figure 7.12: Case Four: GUI Output

Chapter 8

Outlook and Conclusion

The thesis goal can be separated in two parts: firstly the proposal for a hospital localisation system based on the state-of-the-art techniques and secondly the implementation of this proposal. Part one has been succcefully fulfilled. Though it was not possible to implement the full localisation system. The problems faced were that the planned cooperation with the Austrian Academy of Sciences was not able during the thesis. That is why there was no possibility to access the hardware developed by them, which should have been used as base stations after some reprogramming work. Regarding the mobile client it turned out that the available hardware was not sufficient to reach the accuracy needed for a proper localisation. Therefore, the focus has been set on the development of a server software that is able to receive UDP localisation frames and calculate the mobile clients position based on those information. This software has been developed and successfully tested as described in chapter 6 and 7. It uses a trilateration algorithm which calculates the intersection points of the circles drawn around the base stations in the distance of the measurement of the signal propagation delay. By calculating the centroid of the triangle formed by the three shortest lines between the intersection points the clients position is received. For visualising the client in a certain room a display service software has been developed that could run on any computer in the network that is able to connect to the database, that contains the calculated position information.

Next to this results, the thesis provides an overview of the state-of-the-art techniques and summarizes the problems one is faced when trying to set up a localisation system. Because of its importance for the whole process, the time-stamping has been treated with special care.

Additionally the thesis has shown that, due to its simplicity, the round trip time approach is a good alternative to the TDoA method, especially when it comes to special use cases (e.g. the treated hospital use case) where only a special group of persons have to be located. Because in those cases there is no need to integrate the localisation functionality in an existing device like a mobile phone. Instead it could be implemented in an special device which only provides the localisation or as an integration in a personal health monitoring device that collects vital data from several body sensors. Only in the second case the full potential would be used by enabling automatic emergency calls if a vital sign is reaching a critical level.

Another important topic for future research could be to figure out possibilities to mitigate the influences from the environment. Especially in non-line-of-sight environments, which most certainly occur in real world scenarios, the influence on the measurement through reflection and distraction at obstacles, placed in the line between mobile client and base station, are immense. Finding ways to enable a localisation even with heavy obstacles like walls would heavily increase the acceptance, since not every room would needed to be equipped with three antennas. Moreover, the full range of the antennas could be used to cover a certain area, which would heavily decrease the investment costs, since only a fraction of base stations is needed.

Last point for further development is the trilteration algorithm. The developed one lacks functionalities that makes it more fault tolerant and let it work for example even if not all three distances from the base stations are overlapping as described in chapter 5.

Summarising it can be said that till now, it is not possible to use any commercial-of-the-shelf hardware, for a mobile client localisation as described during the thesis. Therefore, these functionalities will take several more years to reach the market and even then it will be a product for special applications only. But since the rising healthcare costs described in chapter 1 are calling for solutions it will be worth to stick to solutions like the ubiquitous patient monitoring that on the other hand implies a localisation of patients in emergency cases.

Bibliography

[1] The vanishing workforce. *The Economist (Online)*, 24.02.2012.

[2] The World Bank. World Development Indicators (WDI) & Global Development Finance (GDF). *World dataBank (databank.worldbank.org)*, 15.12.2011.

[3] Aneeq Mahmood et al. Clock Synchronization in Wireless LANs without Hardware Support. *8th IEEE International Workshop on Factory Communication Systems (WFCS)*, 2010.

[4] Anetta Nagy et al. Time-based Localisation in Unsynchronized Wireless LAN for Industrial Automation Systems. *IEEE ETFA'2011*, 2011.

[5] Dimitrios Lymberopoulos et al. An Empirical Characterization of Radio Signal Strength Variability in 3-D IEEE 802.15.4 Networks Using Monopole Antennas. *Springer-Verlag Berlin Heidelberg*, 2006.

[6] Francesco Potorti et al. Accuracy limits of in-room localisation using RSSI.

[7] Georg Gaderer et al. Localisation of Wireless LAN Nodes using Accurate TDoA Measurements. *WCNC 2010, IEEE Wireless Communications and Networking Conference*, 18.04.2010.

[8] Georg Gaderer et al. Localisation in Wireless Sensor Networks. *Proceedings of the IEEE 2009 Sensors Conference*, 27.10.2009.

[9] Georg Garderer et al. Master Failures in the Precision Time Protocol. *ISPCS 2008 – International IEEE Symposium on Precision Clock Synchronization for Measurement, Control and Communication*, 22.09.2008.

[10] Guoqiang Mao et al. Wireless Sensor Network Localization Techniques.

[11] Hui Liu et al. Survey of Wireless Indoor Positioning Techniques and Systems. *IEEE Transactions on Systems, Man, and Cybernetics*, 06.11.2007.

[12] Kim Sanhae et al. An Improved TDoA-based Tracking Algorithm in Mobile-WiMAX Systems. *IEEE*, 2009.

[13] Luis Mengual et al. Clustering-based location in wireless networks. *Expert Systems with Applications 37 (2010)*, 2010.

[14] M. Ciurana et al. Tracking mobile targets indoors using WLAN and time of arrival. *Computer Communications 32 (2009)*, 2009.

[15] M. Ciurana et al. A robust to multi-path ranging technique over IEEE 802.11 networks. *Wireless Netw (2010)*, 29.04.2009.

[16] Patrick Loschmidt et al. Limits of Synchronization Accuracy Using Hardware Support in IEEE 1588. *ISPCS 2008 – International IEEE Symposium on Precision Clock Synchronization for Measurement, Control and Communication*, 22.09.2008.

[17] Qiuxia Chen et al. Rule-Based WiFi Localization Methods. *2008 IEEE/IFIP International Conference on Embedded and Ubiquitous Computing*, 2008.

[18] Stefan Schwalowsky et al. System Integration of an IEEE 802.11 based TDoA Localization System. *ISPCS 2010 – International IEEE Symposium on Precision Clock Synchronization for Measurement, Control and Communication*, 2010.

[19] invention.smithsonian.org. The Quartz Watch. *http://invention.smithsonian.org*, 06.12.2011.

[20] Alan Kaminsky. Trilateration. 2007.

[21] Harry B. Lee. A Novel Procedure for Assessing the Accuracy of Hyperbolic Multilateration Systems. *IEEE Transactions on Aerospace and Electronic Systems*, 01.1975.

[22] Partrick Loschmidt. On Enhanced Clock Synchronization Performance Through Dedicated Ethernet Hardware Support. *Dissertation*, 2010.

[23] Nimal Nissanke. *Realtime Systems.* Prentice Hall, 1997.

[24] Kaveh Pahlavan. Indoor Geolocation Science and Technology. *IEEE Communications Magazine*, 02.2002.

[25] Sweta Sneha and Upkar Varshney. Enabling ubiquitous patient monitoring: Model, decision protocols, opportunities and challenges. *Elsevier B.V.*, 21.10.2008.

[26] Don J. Torrieri. Statistical Theory of Passive Location Systems. *IEEE Transactions on Aerospace and Electronic Systems*, 03.1984.

[27] IEEE 1588 working group. IEEE Std 1588-2008. *IEEE Standards*, 01.08.2005.

[28] IEEE 802.11 working group. IEEE Std 802.11, 2003 edition. *IEEE Standards*, 01.08.2005.

Appendix A

Java Code: Client

The client code that is presented at the following pages is an autonomous piece of software that simulates the three base stations, which have to send their localisation frames to the server. For trying simulating different situations the user is able to define the distance from the client to each base station by enter the time of flight in nanoseconds before the program sends the frames via UDP to the server. One nanosecond of time of flight corresponds to 28 centimeters of traveled distance. Furthermore the user has to specify a localisation ID which has to be unique for one client localisation process.

C:/Users/Malte Schmidt/Documents/NetBeansProjects/Client/src/client/Client.java

```java

package client;

import java.io.*;
import java.net.*;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;
import java.util.*;
import java.util.logging.Level;
import java.util.logging.Logger;
import org.apache.commons.math3.analysis.function.*;

public class Client {
    public static void main(String[] args) throws IOException {


        String strLocID = javax.swing.JOptionPane.showInputDialog(
                "Please enter the localisation ID.");
        int locID = Integer.parseInt(strLocID);
        String distanceBS1 = javax.swing.JOptionPane.showInputDialog(
                "Please enter the "
                + "distance from BS1 in nanoseconds (1ns = 28cm).");
        String distanceBS2 = javax.swing.JOptionPane.showInputDialog(
                "Please enter the "
                + "distance from BS2 in nanoseconds (1ns = 28cm).");
        String distanceBS3 = javax.swing.JOptionPane.showInputDialog(
                "Please enter the "
                + "distance from BS3 in nanoseconds (1ns = 28cm).");


        Client BS1 = new Client();
        Client BS2 = new Client();
        Client BS3 = new Client();

        BS1.sendLocalPacket("#loc#" + locID + "#1001#2001#283:645:000"
                + "#283:645:"+ distanceBS1 +"##");  //21ns

        BS2.sendLocalPacket("#loc#" + locID + "#1002#2001#283:645:000"
                + "#283:645:"+ distanceBS2 +"##");  //18ns

        BS3.sendLocalPacket("#loc#" + locID + "#1003#2001#283:645:000"
                + "#283:645:"+ distanceBS3 +"##");  //25ns

```

1.1 of 4 2012.05.22 12:50:45

```

    }


    public void sendLocalPacket(String PacketData) throws IOException {

        String hostAddress = "localhost";
        String data = PacketData;
        byte[] response = new byte[256];

            // Create a datagram socket
        DatagramSocket socket = new DatagramSocket();

            // Send the data to the server
        byte[] buf = data.getBytes();
        InetAddress address = InetAddress.getByName(hostAddress);
        DatagramPacket packet = new DatagramPacket(buf, buf.length,
                address, 4445);
        socket.send(packet);

            // get response
        packet = new DatagramPacket(response, response.length);
        socket.receive(packet);

    // display response
        String received = new String(packet.getData());
        System.out.println("Answer from Server: " + received);

        socket.close();

    }




//*******************************************************************
    //* Name: ExecQry
*
    //* Input: SQL Query as a String
*
    //* Output: String Array containing the ReseultSet Data
*
    //* Functionality: Returns the ResultSet Data in form of a Multi-
*
    //*                Dimensional Array (according to the RS)
```

C:/Users/Malte Schmidt/Documents/NetBeansProjects/Client/src/client/Client.java

```
*

//*******************************************************************
    public String[][] ExecQry(String input){

        Connection conn = null;
        String url = "jdbc:mysql://localhost/mydb";
        String user = "root";
        String pass = "malte";
        String strSQL = input;
        ResultSet rs = null;
        String[][] stringResult = null;
        int NoRows = 0;
        int NoColumns = 0;
        int ArrayCount = 0;
        int RSCount = 1;

        try {
                Class.forName("com.mysql.jdbc.Driver").newInstance();
                conn = DriverManager.getConnection(url, user, pass);
                System.out.println("PosCalcThread: ExecQry: "
                        + "Connected to DB");
                Statement stmt = conn.createStatement();
                rs = stmt.executeQuery(strSQL);

                    //Getting Number of Rows
                rs.last();
                NoRows = rs.getRow();
                rs.first();

                    //Getting Number of Columns
                NoColumns = rs.getMetaData().getColumnCount();

                    //Initializing the string-Array
                stringResult = new String[NoRows][NoColumns];

                    //Writing ResultSet Data in string-Array
                for(int i = 0; i < NoRows; i++){
                    for(int j = 0; j < NoColumns; j++){
                        stringResult[i][j] = rs.getString(j+1);
                    }
                    rs.next();
                }

            }
```

3.1 of 4 2012.05.22 12:50:45

C:/Users/Malte Schmidt/Documents/NetBeansProjects/Client/src/client/Client.java

```
            catch (Exception e) {
                System.err.println("PosCalcThread: ExecQry: "
                        + "Connection poblem:");
                e.printStackTrace();
            }
            finally {
                if (conn != null) {
                    try {
                        conn.close();
                        System.out.println("PosCalcThread: ExecQry: "
                                + "Connection closed.");
                    } catch(Exception e) {
                        System.out.println("PosCalcThread: ExecQry: "
                                + "Failure while closing connection:");
                        e.printStackTrace();
                    }
                }
            }

        return stringResult;
    }

}
```

4.1 of 4 2012.05.22 12:50:45

Appendix B

Java Code: Server

The following sections are containing the program code of the server software. An overview is illustrated in figure B.1.

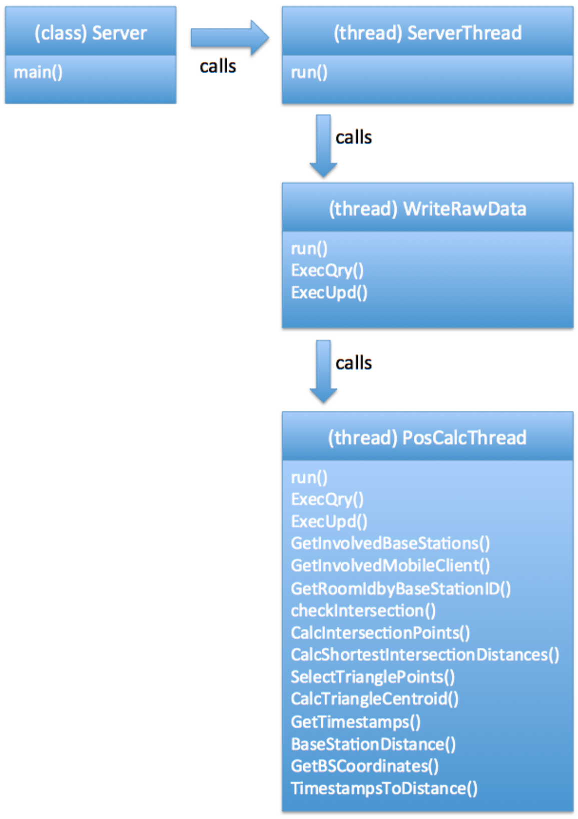


Figure B.1: The Server Classes

B.1 Server

The next page contains the server class which simply starts the server thread from its main class.

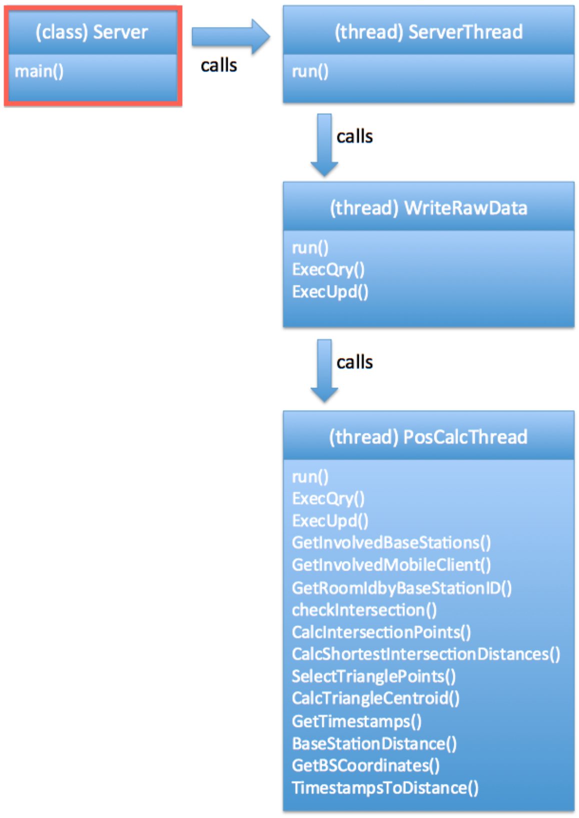


Figure B.2: The Server Classes

C:/Users/Malte Schmidt/Documents/NetBeansProjects/Server/src/server/Server.java

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package server;

import java.io.*;


public class Server{

    public static void main(String[] args) throws IOException {

        new ServerThread().start();


    }

}
```

1.1 of 1 2012.05.21 17:17:57

B.2 ServerThread

The following contained server thread is an infinite loop that is listening for new packets from the mobile client and starts the subsequent processing stages: writing the raw data to the database and calculate the position of the mobile client.

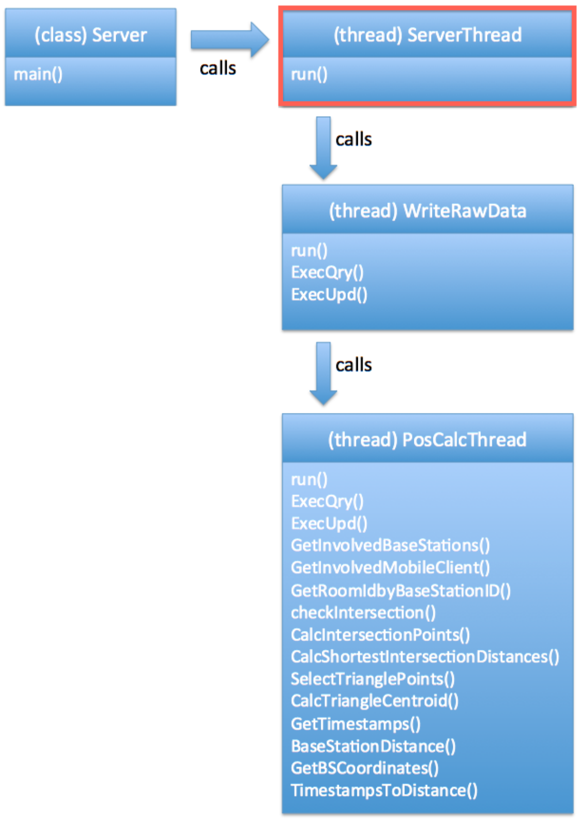


Figure B.3: The Server Classes

C:/Users/Malte Schmidt/Documents/NetBeansProjects/Server/src/server/ServerThread.java

```java
package server;

import java.io.*;
import java.net.*;
import java.util.*;
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;
import java.util.logging.Level;
import java.util.logging.Logger;

public class ServerThread extends Thread {

    protected DatagramSocket socket = null;
    Collection list1 = new LinkedList();
    int NoWRD = 0;
    int PosCalcStart = 0;
    final Lock lock = new ReentrantLock();

      public ServerThread() throws IOException{
          socket = new DatagramSocket(4445);
      }


    public void run() {


        while (true) {
            try {
                byte[] buf = new byte[256];
                String receivedData = new String();
                String responseText = "#PacketReceived_ACK##";
                byte[] response = responseText.getBytes();

                    // receive request
                DatagramPacket packet = new DatagramPacket(buf,
                        buf.length);
                socket.receive(packet);

                receivedData = new String(packet.getData());

                    // show received data
                System.out.printf("ServerThread: Received Packet: "
                        + "Addressed to %s at Port %d with %d Byte, "
                        + "Content:%n%s%n%n",
                            packet.getAddress(), packet.getPort(),
```

1.1 of 3 2012.05.22 12:44:51

```
                              packet.getLength(),
                              new String(packet.getData()) );

                      // send response to the client
                  InetAddress address = packet.getAddress();
                  int port = packet.getPort();
                  packet = new DatagramPacket(response, response.length,
                          address, port);
                  socket.send(packet);

                  System.out.printf("ServerThread: Response Packet: "
                          + "Addressed to %s at Port %d with %d Byte, "
                          + "Content:%n%s%n%n",
                              packet.getAddress(), packet.getPort(),
                              packet.getLength(),
                              new String(packet.getData()) );

                      // splitting the string for getting the Loclisation
ID
                  String LocID = new String();
                  String[] splitted =
                          new String[receivedData.split("#").length];
                  splitted = receivedData.split("#");
                  LocID = splitted[2];


                      // start Thread to write RAW Data in DB
                  lock.lock();
                  if(receivedData.startsWith("#loc#")){
                      new WriteRawData(receivedData).start();
                  }else
                      System.out.printf("%nNo localisation Frame");

                  try {
                      Thread.currentThread().sleep(400);
                  } catch (InterruptedException ex) {
                      Logger.getLogger(ServerThread.class.getName()).log(
                              Level.SEVERE, null, ex);
                  }
                       //start position calculation thread
                       new PosCalcThread(Integer.parseInt(LocID)).start();
                  lock.unlock();

              } catch (IOException e) {
                  System.out.println("A problem occured: ");
```

C:/Users/Malte Schmidt/Documents/NetBeansProjects/Server/src/server/ServerThread.java

```
                e.printStackTrace();
            }
        }
    }


}
```

3.1 of 3 2012.05.22 12:44:51

B.3 WriteRawData

The WriteRawData class is started by the server thread and write the information of the received localisation frames to the database.

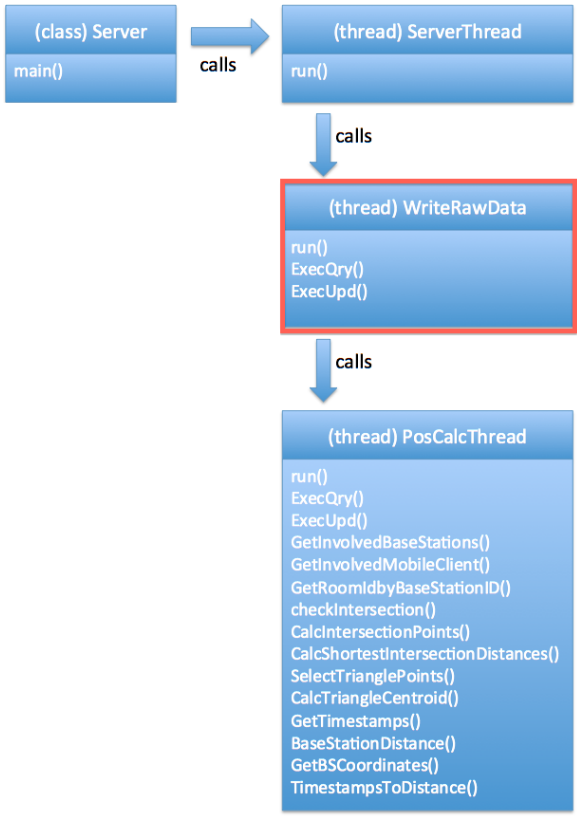


Figure B.4: The Server Classes

C:/Users/Malte Schmidt/Documents/NetBeansProjects/Server/src/server/WriteRawData.java

```

package server;

import java.sql.*;
import java.util.logging.Level;
import java.util.logging.Logger;
import java.util.concurrent.locks.ReentrantLock;
import java.util.concurrent.locks.Lock;

public class WriteRawData extends Thread {

    protected String data = null;
    int PosCalcStart = 0;
    final Lock lock = new ReentrantLock();

    public WriteRawData(String rawData){
        this.data = rawData;
    }


    public void run(){

            String[] rs = null;
            int rsLength = 0;
            int MeasurementValues = 3;
            String[] rs2 = null;

            System.out.printf("%nWrite RAW Data starts...%n");
            System.out.printf("RawData: " + data + "%n%n");

            // splitting the string
            String T1, T2, LocID, BSID, MCID = new String();
            String[] splitted = new String[data.split("#").length];
            splitted = data.split("#");
            LocID = splitted[2];
            BSID = splitted[3];
            MCID = splitted[4];
            T1 = splitted[5];
            T2 = splitted[6];

            // Test if already three BS entrys saved in DB
            String strSQL = "SELECT COUNT(LocalisationID) "
                    + "FROM tblframe "
                    + "WHERE LocalisationID = " +
Integer.parseInt(LocID);
```

1.1 of 5 2012.05.22 12:54:34

C:/Users/Malte Schmidt/Documents/NetBeansProjects/Server/src/server/WriteRawData.java

```java
            rs = ExecQry(strSQL);
            if(rs.length != 0){
                if(Integer.parseInt(rs[0]) < MeasurementValues){
                        strSQL = "INSERT INTO tblframe (LocalisationID,"
                                + "MobileClientID,BaseStationID,"
                                + "OutgoingTime,IncomingTime) "
                                + "VALUES ("
                                + Integer.parseInt(LocID) + ","
                                + Integer.parseInt(MCID) + ","
                                + Integer.parseInt(BSID) + ","
                                + "'" + T1 + "'" + "," + "'" + T2 + "')";
                        ExecUpd(strSQL);

                        System.out.println("WriteRawData: "
                                + "RawData is written");


                }else
                    System.out.println("WriteRawData: "
                            + "Threre are already three "
                            + "entrys for this localisation ID "
                            + "stored in the DB!");
            }else
                System.out.println("WriteRawData: Failure in "
                        + "ResultSet");

    }



//*******************************************************************
    //* Name: ExecUpd                                                    *
    //* Input: SQL String as String                                      *
    //* Output: -                                                        *
    //* Description: Takes an SQL String with and performs the Update on*
    //*              the DataBase                                        *

//*******************************************************************
    public void ExecUpd(String input){

        Connection conn = null;
        String url = "jdbc:mysql://localhost/mydb";
        String user = "root";
        String pass = "malte";
        String strSQL = input;
        ResultSet rs = null;

        try {
                Class.forName("com.mysql.jdbc.Driver").newInstance();
                conn = DriverManager.getConnection(url, user, pass);
                System.out.println("WriteRawData: ExecUpd: "
                        + "Connected to DB");
                Statement stmt = conn.createStatement();
                stmt.executeUpdate(strSQL);
                conn.setAutoCommit(true);
                conn.close();

            }
            catch (Exception e) {
                System.err.println("WriteRawData: ExecUpd: "
                        + "Connection poblem:");
                e.printStackTrace();
            }
            finally {
                if (conn != null) {
                    try {
                        conn.close();
                        System.out.println("WriteRawData: ExecUpd: "
                                + "Connection closed.");

                    } catch(Exception e) {
                        System.out.println("WriteRawData: ExecUpd: "
                                + "Failure while closing connection");
                    }
                }
            }

    }



    //NOTE: Eventually exchange to newer Version like in
PosCalcThread.java
    public String[] ExecQry(String input){

        Connection conn = null;
        String url = "jdbc:mysql://localhost/mydb";
        String user = "root";
        String pass = "malte";
        String strSQL = input;
        ResultSet rs = null;
        String[] stringResult = null;
        int ArrayCount = 0;
        int RSCount = 1;

        try {
                Class.forName("com.mysql.jdbc.Driver").newInstance();
                conn = DriverManager.getConnection(url, user, pass);
                System.out.println("WriteRawData: ExecQry: Connected to
DB");
                Statement stmt = conn.createStatement();
                rs = stmt.executeQuery(strSQL);

                stringResult = new
String[rs.getMetaData().getColumnCount()];

                while(rs.next()){
                    stringResult[ArrayCount] = rs.getString(RSCount);
                    ArrayCount++;
                    RSCount++;
                }

            }
            catch (Exception e) {
                System.err.println("WriteRawData: ExecQry: "
                        + "Connection poblem:");
                e.printStackTrace();
            }
            finally {
                if (conn != null) {
                    try {
                        conn.close();
                        System.out.println("WriteRawData: ExecQry: "
                                + "Connection closed.");
                    } catch(Exception e) {
                        System.out.println("WriteRawData: ExecQry: "
                                + "Failure while closing connection:");
                        e.printStackTrace();
                    }
```

C:/Users/Malte Schmidt/Documents/NetBeansProjects/Server/src/server/WriteRawData.java

```
                }
            }

        return stringResult;
    }




}
```

5.1 of 5 2012.05.22 12:54:34

B.4 PositionCalculationThread

The next pages contain the Java code for the calculation of the mobile clients position, which is by far the most complex class.

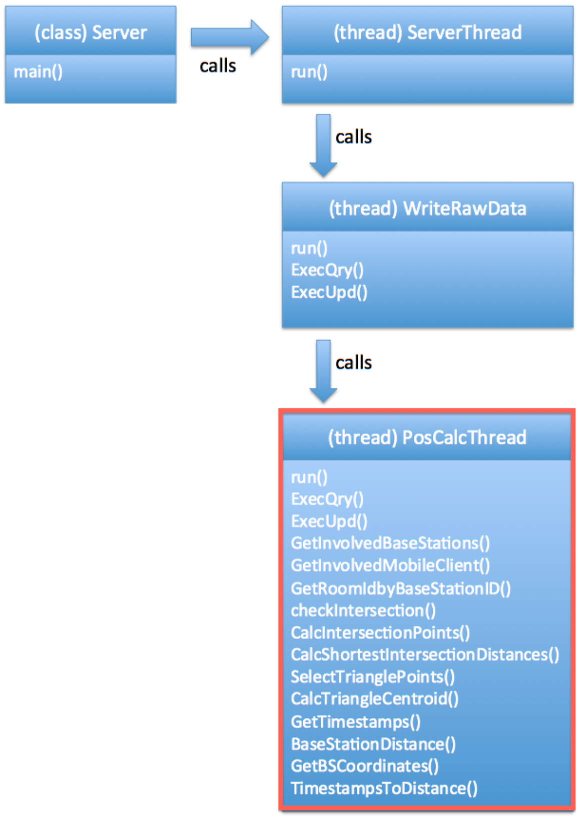


Figure B.5: The Server Classes

C:/Users/Malte Schmidt/Documents/NetBeansProjects/Server/src/server/PosCalcThread.java

```

package server;

import java.sql.*;
import java.math.*;
import java.util.Arrays;
import org.apache.commons.math3.analysis.function.*;
import java.util.*;

public class PosCalcThread extends Thread {

    protected int locID = 0;
    protected int numberOfBaseStations = 3;


    public PosCalcThread(int LocID){
        this.locID = LocID;
    }


    public void run(){

        System.out.printf("%nPosition Calculation for LocID: " + locID
                + " starts%n");

        String[][] rs = null;
        int MeasurementValues = 3;
        String[] baseStationIDs;
        int mobileClientID;
        int RoomID;
        double[] allInterSecPoints = new double[12];
        double[][] shortestInterSecDist;


            // Test if already three BS entrys saved in DB
        String strSQL = "SELECT COUNT(LocalisationID) FROM tblframe "
                + "WHERE LocalisationID = " + locID;
        rs = ExecQry(strSQL);
        if(rs.length != 0){
            if(Integer.parseInt(rs[0][0]) == MeasurementValues){

                System.out.println("PosCalcThread: Calculation starts...
"
                        + "No. of Entrys: " + rs[0][0]);

```

1.1 of 23 2012.05.22 11:14:33

```
                baseStationIDs = GetInvolvedBaseStations(locID);
                mobileClientID = GetInvolvedMobileClient(locID);

                RoomID = GetRoomIDbyBaseStationID(
                        Integer.parseInt(baseStationIDs[0]));


                boolean intersection1 = checkIntersection(locID,
                        Integer.parseInt(baseStationIDs[0]),
                        Integer.parseInt(baseStationIDs[1]));
                boolean intersection2 = checkIntersection(locID,
                        Integer.parseInt(baseStationIDs[1]),
                        Integer.parseInt(baseStationIDs[2]));
                boolean intersection3 = checkIntersection(locID,
                        Integer.parseInt(baseStationIDs[0]),
                        Integer.parseInt(baseStationIDs[2]));

                if(intersection1 && intersection2 && intersection3){

                    double[] ISPoints1;
                    ISPoints1 = CalcIntersectionPoints(locID,
                            Integer.parseInt(baseStationIDs[0]),
                            Integer.parseInt(baseStationIDs[1]));

                    double[] ISPoints2;
                    ISPoints2 = CalcIntersectionPoints(locID,
                            Integer.parseInt(baseStationIDs[1]),
                            Integer.parseInt(baseStationIDs[2]));

                    double[] ISPoints3;
                    ISPoints3 = CalcIntersectionPoints(locID,
                            Integer.parseInt(baseStationIDs[0]),
                            Integer.parseInt(baseStationIDs[2]));


                    allInterSecPoints[0] = ISPoints1[0];
                    allInterSecPoints[1] = ISPoints1[1];
                    allInterSecPoints[2] = ISPoints1[2];
                    allInterSecPoints[3] = ISPoints1[3];
                    allInterSecPoints[4] = ISPoints2[0];
                    allInterSecPoints[5] = ISPoints2[1];
                    allInterSecPoints[6] = ISPoints2[2];
                    allInterSecPoints[7] = ISPoints2[3];
                    allInterSecPoints[8] = ISPoints3[0];
                    allInterSecPoints[9] = ISPoints3[1];
                    allInterSecPoints[10] = ISPoints3[2];
                    allInterSecPoints[11] = ISPoints3[3];

                    shortestInterSecDist =
CalcShortestIntersectionDistances(allInterSecPoints);

                    System.out.printf("%n%nShortest Distances and
according"
                            + " x-y- coordinates:%n");
                    for(int i = 0; i < 3; i++){
                        for(int j = 0; j < 5; j++){
                            System.out.print(shortestInterSecDist[i][j]
                                    + " ");
                        }
                        System.out.println("");
                    }

                    double[] centroid = CalcTriangleCentroid(
                            SelectTrianglePoints(shortestInterSecDist));

                    System.out.printf("%n%nCentroid coordinates:%n%n");
                    System.out.println("Cx: " + centroid[0]);
                    System.out.println("Cy: " + centroid[1]);

                        // Write calculated position data in DB
                    strSQL = "INSERT INTO tblposition (MobileClientID, "
                            + "Date, Time, RoomID, RoomRelPosX, "
                            + "RoomRelPosY, LocalisationID) "
                            + "VALUES ("
                                    + mobileClientID + ","
                                    + " CURRENT_DATE(),"
                                    + " NOW(),"
                                    + RoomID + ","
                                    + centroid[0] + ","
                                    + centroid[1] + ","
                                    + locID + ")";
                            ExecUpd(strSQL);
                }else
                    System.out.println("The circles do not intersect!");

            }else
                System.out.println("PosCalcThread: Cant start, now.
Threre "
                        + "are only " + rs[0][0] + "entrys for this "
```

```
                            + "localisation ID stored in the DB.");
            }else
                System.out.println("PosCalcThread: Failure in ResultSet");


        }


//******************************************************************
        //* Name: ExecUpd
*
        //* Input: SQL String as String
*
        //* Output: -
*
        //* Description: Takes an SQL String with and performs the Update
on*
        //*             the DataBase
*

//******************************************************************
        public void ExecUpd(String input){

            Connection conn = null;
            String url = "jdbc:mysql://localhost/mydb";
            String user = "root";
            String pass = "malte";
            String strSQL = input;
            ResultSet rs = null;

            try {
                        // Settting the driver and creating a connection
                    Class.forName("com.mysql.jdbc.Driver").newInstance();
                    conn = DriverManager.getConnection(url, user, pass);
                        // Creating a statement object and exectuting the
update
                    Statement stmt = conn.createStatement();
                    stmt.executeUpdate(strSQL);
                        // AutoCommit the Update and close the connection
                    conn.setAutoCommit(true);
                    conn.close();

                }
                catch (Exception e) {
                    System.err.println("PosCalcThread: ExecUpd: "
                            + "Connection poblem:");
                    e.printStackTrace();
                }


                finally {
                    if (conn != null) {
                        try {
                            conn.close();
                            System.out.println("PosCalcThread: ExecUpd: "
                                    + "Connection closed.");

                        } catch(Exception e) {
                            System.out.println("PosCalcThread: ExecUpd: "
                                    + "Failure while closing connection");
                        }
                    }
                }

        }


//*****************************************************************
        //* Name: GetRoomIDbyBaseStationID
*
        //* Input: BaseStation ID as integer
*
        //* Output: Room ID as integer
*
        //* Description: Retrieves the ID of the room, where a specific
base*
        //*              station is installed in
*

//*****************************************************************
        public int GetRoomIDbyBaseStationID (int BaseStationID){

            String[][] rs;
            int RoomID;
            String strSQL = "SELECT RoomID"
                    + " FROM tblbasestation"
                    + " WHERE idtblbasestation = " + BaseStationID;

            rs = ExecQry(strSQL);
            RoomID = Integer.parseInt(rs[0][0]);

            return RoomID;
        }



//*****************************************************************
        //* Name: CalcTriangleCentroids
*
        //* Input: Three x-y point coordinates of a triangle as a double
*
        //*         array
*
        //* Output: x-y-coordinates as double array
*
        //*         output[0] = x-coordinate
*
        //*         output[1] = y-coordinate
*
        //* Description: Calculates the centroid coordinates of a given
*
        //*              triangle
*
        //*
*
//*****************************************************************
        public double[] CalcTriangleCentroid(double[][] TrianglePoints){

            double[] centroidCoordinates = new double[2];

            double P1x, P1y, P2x, P2y, P3x, P3y;
            double Cx, Cy;
            Divide div = new Divide();

            P1x = TrianglePoints[0][0];
            P1y = TrianglePoints[0][1];

            P2x = TrianglePoints[1][0];
            P2y = TrianglePoints[1][1];

            P3x = TrianglePoints[2][0];
            P3y = TrianglePoints[2][1];

            Cx = div.value(1, 3) * (P1x + P2x + P3x);
```

```java
        Cy = div.value(1, 3) * (P1y + P2y + P3y);

        centroidCoordinates[0] = Cx;
        centroidCoordinates[1] = Cy;

        return centroidCoordinates;
    }


    //*****************************************************************
    //* Name: SelectTrianglePoints                                    *
    //* Input: 3 Distances and their start and end x-y-coordinates as a *
    //*        double array.                                          *
    //* Output: 3 x-y-coordinates as a double array                   *
    //* Description: Selects the three points which are bulding a     *
    //*              triangle out of the six start and end points     *
    //*              (given as x-y-coordinates) defining the three    *
    //*              lines of the triangle                            *
    //*****************************************************************
    public double[][] SelectTrianglePoints(double[][] Points){

        double[] workSet = new double[12];
        double[] buffer = new double[6];
        double x,y;
        double[][] TrianglePoints = new double[3][2];
        int counter = 0;

            // Writing the input data into a one-dimensional workset array
        for(int i = 0; i < 3; i++){
            for(int j = 0; j < 4; j++){
                workSet[counter] = Points[i][j];
                counter++;
            }
        }

        System.out.println("WorkSet:");
        for(int i = 0; i < 12; i++){
            System.out.print(workSet[i] + ", ");
        }
        System.out.printf("%n%n");

        int nextFree = 0;
            // Elemenating equal points from the workSet by going 
through
            // the workset array and writing the cointaining
            // x-y-coordiantes into the buffer array if not already in.
            // Target: Elemenating the double contained points which 
defines
            // the lines between the points of the triangle
        for(int i = 0; i < (workSet.length-1); i = i + 2){
            x = workSet[i];
            y = workSet[i+1];
            System.out.println("x = " + workSet[i]);
            System.out.println("y = " + workSet[i+1]);
            System.out.println(x != buffer[0]);
            if(((x != buffer[0]) || (y != buffer[1])) &&
               ((x != buffer[2]) || (y != buffer[3])) &&
               ((x != buffer[4]) || (y != buffer[5]))){
                    System.out.println("NextFree: " + nextFree);
                    buffer[nextFree] = x;
                    System.out.println("buffer " + nextFree+ ": "
                            + buffer[nextFree]);
                    buffer[nextFree+1] = y;
                    System.out.println("buffer " + nextFree+1+ ": "
                            + buffer[nextFree+1]);
                    nextFree = nextFree + 2;
                    System.out.println("NextFree: " + nextFree);

            }
        }

            // Print the buffer Array to screen
        System.out.printf("%n%nBuffer Array:%n");
        for(int j = 0; j < buffer.length; j++){
            System.out.println(buffer[j]);
        }

            //Putting Elements from buffer to Array TrianglePoints
        counter = 0;
        for (int i = 0; i < 3; i++){
            for (int j = 0; j < 2; j++){
                TrianglePoints[i][j] = buffer[counter];
                counter++;
            }
        }

        return TrianglePoints;
    }



//*****************************************************************
    //* Name: CalcShortestIntersectionDistances                                *
    //* Input: double Array containing of all 12 intersection points          *
    //* Output: two-dimensional array containing the distance of the          *
    //*         three shortest lines between intersection points as           *
    //*         well as the x-y-coordinates defining start and end of         *
    //*         these lines                                                    *
    //*         output[0][0]  [0][1]    [0][2]      [0][3]      [0][4]         *
    //*         x-coordinate  y-coord.  x-coord.    x-coord.   Distance        *
    //*         start point   start p.  end-point   end-p.                     *
    //* Description: Calculating the shortest lines between the six           *
    //*              intersection points                                       *
//*****************************************************************
    public double[][] CalcShortestIntersectionDistances(double[] ISPoints){

        double[][] output = new double[3][5];
        double Dx, Dy, D;

        double[] Intersections1 = ISPoints;
        double[] Intersections2 = ISPoints;
        double[][] Distances = new double[6][6];
        double[][] ISDistances = new double[15][5];

        int i = 0;
        int j = 2;
        int d = 1;
        int e = 0;
        int count = 0;


            // Calculate the distances of all needed lines between the
            // 6 different intersection points and save them into two
            // different Arrays: one just saving the distances and the
            // second saving the distances as well as the according
            // x-y-coordinates
        while(i < Intersections1.length){
            while(j < Intersections2.length){

                Dx = Intersections2[j] - Intersections1[i];
                Dy = Intersections2[j+1] - Intersections1[i+1];

                D = Math.sqrt(Math.pow(Dx,2) + Math.pow(Dy,2));
                Distances[e][d] = D;

                ISDistances[count][0] = Intersections1[i];
                ISDistances[count][1] = Intersections1[i+1];
                ISDistances[count][2] = Intersections2[j];
                ISDistances[count][3] = Intersections1[j+1];
                ISDistances[count][4] = D;

                count++;
                j = j+2;
                d++;
            }
            i = i+2;
            j = i+2;
            e++;
            d = e+1;
        }

            // Printing the Array just saving the distances on the
screen
        for(int k = 0; k < Distances.length; k++){
            for(int l = 0; l < Distances.length; l++){
                System.out.print(Distances[k][l] + "  ");
```

```java
            }
            System.out.println("");
        }

            // Printing the Array containing the distances and the
            // x-y-coordinates on the screen
        System.out.println("-----------------------------------------");
        for(int k = 0; k < 15; k++){
            for(int l = 0; l < 5; l++){
                System.out.print(ISDistances[k][l] + "  ");
            }
            System.out.println("");
        }
        System.out.println("------------------------------------------");


            // Sorting the before printed Array by using a reference Array
            // by the distance of the lines. After sorting, the first three
            // entrys are the searched ones with the shortest distances.
        int[] RefArray = {0,1,2,3,4,5,6,7,8,9,10,11,12,13,14};
        boolean done = false;
        int doneCount;
        int x = 0, y = 0;
        int ref1, ref2;

        System.out.println("Start sorting the RefArray");
        while(done == false){
            doneCount = 0;
            for(i = 0; i < 14; i++){
                if(ISDistances[RefArray[i]][4]
                        > ISDistances[RefArray[i+1]][4]){
                    x = RefArray[i];
                    y = RefArray[i+1];
                    RefArray[i] = y;
                    RefArray[i+1] = x;
                    doneCount++;
                }
            }
            if(doneCount == 0)
                done = true;
        }
```

```

            // Printing the sorted reference Array
        System.out.println("Sorted RefArray: ");
        for(i = 0; i < 15; i++)
            System.out.print(", " + RefArray[i]);

        System.out.println("");

            // Writing the first three (searched) entrys of the Array
            // containing the distances as well as the point coordinates
            // into the output array.
        for(i = 0; i < 3; i++){
            for(j = 0; j < 5; j++){
                output[i][j] = ISDistances[RefArray[i]][j];
            }
        }


        return output;
    }



//******************************************************************
    //* Name: CalcIntersectionPoints
*
    //* Input: Localisation ID and two Base Station IDs as Integer
*
    //* Output: x-y-coordinates of the two intersection points (IS)
*
    //*         output[0] : x-coordinate of IS1
*
    //*         output[1] : y-coordinate of IS1
*
    //*         output[2] : x-coordinate of IS2
*
    //*         output[3] : y-coordinate of IS2
*
    //* Description: Calculates the intersection points of the circles,
*
    //*              drawn by the distances to the MC around the BS
*

//******************************************************************
    public double[] CalcIntersectionPoints(int localisationID, int
BS1ID,
            int BS2ID){

        double[] output = new double[4];
        float RadiusBS1, RadiusBS2;
        double S1, S2, S3;
        double BSAx, BSAy, BSBx, BSBy, Dx, Dy, D;
        double[] coordinates;
        String[] TimestampsBS1, TimestampsBS2;
        double IS1x, IS1y, IS2x, IS2y, Cx, Cy;

            // Get coordinates according to BaseStation ID
        coordinates = GetBSCoordinates(BS1ID);
        BSAx = coordinates[0];
        BSAy = coordinates[1];

            //Get coordinates according to BaseStation ID
        coordinates = GetBSCoordinates(BS2ID);
        BSBx = coordinates[0];
        BSBy = coordinates[1];

            // Calculate the Distance between the BaseSations
        Dx = BSBx - BSAx;
        Dy = BSBy - BSAy;
        D = Math.sqrt(Math.pow(Dx,2) + Math.pow(Dy, 2));


            // Get the timestamps of BS1
        TimestampsBS1 = GetTimestamps(locID, BS1ID);

            // Get the radius of BS1
        RadiusBS1 = TimestampsToDistance(TimestampsBS1);


            // Get the timestamps of BS2
        TimestampsBS2 = GetTimestamps(locID, BS2ID);

            // Get the radius of BS1
        RadiusBS2 = TimestampsToDistance(TimestampsBS2);

            // Initialising object for being able to devide two numbers
        Divide d = new Divide();

           // Calculating the distance to the point which is on the line
           // between both base stations and on the line between both
```

```java
            // intersection points
        S1 = d.value((Math.pow(D, 2) + Math.pow(RadiusBS1, 2)
                - Math.pow(RadiusBS2, 2)), 2*D) ;

            // Calculating the x-y-coordinates of the point which is on
            // the line between both base stations and on the line between
            // both intersection points (Point C)
        Cx = BSAx + d.value((Dx * S1), D);
        Cy = BSAy + d.value((Dy * S1), D);

            // Calculating the distance from point C to the
            // intersection points)
        S3 = Math.sqrt(Math.pow(RadiusBS1, 2) - Math.pow(S1, 2));

            // Calculate the x-y-coordinates of the first
            // intersection point
        IS1x = Cx - d.value((Dy * S3), D);
        IS1y = Cy + d.value((Dx * S3), D);

           // Calculate the x-y-coordinates of the second
           // intersection point
        IS2x = Cx + d.value((Dy * S3), D);
        IS2y = Cy - d.value((Dx * S3), D);

           // Write the x-y-coordinates of the intersection points
           // into an array
        output[0] = IS1x;
        output[1] = IS1y;
        output[2] = IS2x;
        output[3] = IS2y;

        return output;
    }



//*********************************************************************
    //* Name: GetTimestamps                                          *
    //* Input: Localisation ID and BaseStation ID as String Array    *
    //* Output: Timestamps according to LocID and BS ID              *
```

C:/Users/Malte Schmidt/Documents/NetBeansProjects/Server/src/server/PosCalcThread.java

```
    //* Description: Gets the Timestamps of one measurement for one
*
    //*              Localisation ID and one Base Station from the DB
*
//*******************************************************************
    public String[] GetTimestamps(int localisationID, int
BaseStationID){

        String[] output = new String[2];
        int locID = localisationID;
        int BSID = BaseStationID;
        int NoRows, NoColumns;
        String[][] rs;
        String strSQL = "SELECT OutgoingTime, IncomingTime"
                + " FROM tblframe"
                + " WHERE LocalisationID = " + locID
                + " AND BaseStationID = " + BSID;
        rs = ExecQry(strSQL);

            //Getting number of rows and columns
        NoRows = rs.length;
        NoColumns = rs[0].length;

            // Getting Base Station IDs from ResultSet string-Array
            // NOTE: Check to replace with System.arraycopy
        for(int i = 0; i < NoRows; i++){
            for(int j = 0; j < NoColumns; j++){
                output[j] = rs[i][j];
            }
        }

        return output;
    }


    //*****************************************************************
    //* Name: checkIntersection                                       *
    //* Input: A localisation ID and two Base Station IDs             *
    //* Output: true&false as boolean                                 *
    //* Functionality: Calculates if the circles of a certain         *
    //*                measurement around two base stations are       *
    //*                intersecting each other                        *
    //*****************************************************************
    public boolean checkIntersection(int localisationID, int BSID1,
            int BSID2){

        int locID = localisationID;
        int BS1 = BSID1;
        int BS2 = BSID2;
        String[] BaseStations = new String[2];
       // BaseStations[0] = BS1;
        //BaseStations[1] = BS2;
        boolean output;
        double BSDistance;
        float RadiusBS1, RadiusBS2;
        String[] TimestampsBS1, TimestampsBS2;

            // Get the distance between the both BS
        BSDistance = BaseStationDistance(BS1, BS2);


            // Get the timestamps of BS1
        TimestampsBS1 = GetTimestamps(locID, BS1);

            // Get the radius of BS1
        RadiusBS1 = TimestampsToDistance(TimestampsBS1);


            // Get the timestamps of BS2
        TimestampsBS2 = GetTimestamps(locID, BS2);

            // Get the radius of BS1
        RadiusBS2 = TimestampsToDistance(TimestampsBS2);

            // Check, if the both radius are intersecting
        if(BSDistance > (RadiusBS1 + RadiusBS2))
            output = false;
        else
            output = true;


        return output;
    }



//*********************************************************************
    //* Name: BaseStationDistance
*
```

```
    //* Input: Two Base Statio IDS as a String Array *
    //* Output: The Distance between the both Base Stations as float *
    //* Functionality: Calcultes the Distance between two BS *
    //* *
//*******************************************************************
    public double BaseStationDistance(int BaseStationA, int BaseStationB){

        int BSA = BaseStationA;
        int BSB = BaseStationB;
        double[] coordinates;

        double BSAx, BSAy, BSBx, BSBy, Dx, Dy, D = 0;

        coordinates = GetBSCoordinates(BSA);
        BSAx = coordinates[0];
        BSAy = coordinates[1];

        coordinates = GetBSCoordinates(BSB);
        BSBx = coordinates[0];
        BSBy = coordinates[1];

        Dx = BSBx - BSAx;
        Dy = BSBy - BSAy;
        D = Math.sqrt(Math.pow(Dx,2) + Math.pow(Dy, 2));

        return D;
    }



//*******************************************************************
    //* Name: GetBSCoordinates *
    //* Input: Base Station ID as String *
    //* Output: X-Y-Coordinates of the BS as a float Array *
    //* Description: Gets the X-Y-Coordinates of the Base Station from *
    //*              the according table of the DB
*
    //* float coordinates[0] = x-coordinate
*
    //* float coordinates[1] = y-coordinate
*

//*******************************************************************
    public double[] GetBSCoordinates(int BaseStationID){

        int BSID = BaseStationID;

        String[][] rs;
        int count = 0;
        int NoRows = 0;
        int NoColumns = 0;
        double[] coordinates = new double[2];

        String strSQL = "SELECT RoomRelPosX, RoomRelPosY "
                + "FROM tblbasestation "
                + "WHERE idtblbasestation = " + BSID;
        rs = ExecQry(strSQL);

            //Getting number of rows and columns
        NoRows = rs.length;
        NoColumns = rs[0].length;

            //Getting Base Station IDs from ResultSet string-Array
        for(int i = 0; i < NoRows; i++){
            for(int j = 0; j < NoColumns; j++){
                coordinates[j] = Double.parseDouble(rs[i][j]);
            }
        }

        return coordinates;
    }




//*******************************************************************
    //* Name: GetInvolvedMobileClient
*
    //* Input: Localisation ID as Integer
*
    //* Output: Mobile Client ID as Integer
```

```
*
    //* Description: Gets the Mobile Client ID according to the
*
    //*                localisation ID from the DB
*

//*********************************************************************
    public int GetInvolvedMobileClient(int LocalID){

        int MobileClient;
        String[][] rs;
        int count = 0;
        int NoRows = 0;
        int NoColumns = 0;

        String strSQL = "SELECT MobileClientID "
                + "FROM tblframe WHERE "
                + "LocalisationID = " + LocalID;
        rs = ExecQry(strSQL);

        MobileClient = Integer.parseInt(rs[0][0]);

        return MobileClient;
    }



//*********************************************************************
    //* Name: GetInvolvedBaseStations
*
    //* Input: Localisation ID as Integer
*
    //* Output: Localisation Involved Base Stations as a String Array
*
    //* Description: Gets the Base Stations involved in the
Localisation*
    //*              according to the localisation ID from the DB
*

//*********************************************************************
    public String[] GetInvolvedBaseStations(int LocalID){

        String[] BaseStations = new String[numberOfBaseStations];
        String[][] rs;
        int count = 0;
        int NoRows = 0;
        int NoColumns = 0;

        String strSQL = "SELECT BaseStationID "
                + "FROM tblframe WHERE "
                + "LocalisationID = " + LocalID;
        rs = ExecQry(strSQL);

            //Getting number of rows and columns
        NoRows = rs.length;
        NoColumns = rs[0].length;

            //Getting Base Station IDs from ResultSet string-Array
        for(int i = 0; i < NoRows; i++){
            for(int j = 0; j < NoColumns; j++){
                BaseStations[i] = rs[i][j];
            }
        }

        System.out.println("Involved Base Stations: " + BaseStations[0]
                + ", " + BaseStations[1] + ", " + BaseStations[2]);

        return BaseStations;
    }


//*******************************************************************
    //* Name: TimestampToDistance
*
    //* Input: Two Timestamps recorded at one BS as a String Array
*
    //* Output: The distance in meters between the BS and the MC
*
    //* Functionality: Calculates the distance according to the
*
    //*                Timestamps
*

//*******************************************************************
    public float TimestampsToDistance(String[] Timestamps) {

        float VOP = (float) 0.28; // Velocity of Propagatio
                                  // (~28cm/nanosecond)
        String T1 = Timestamps[0];
        String T2 = Timestamps[1];
        String[] T1Splitted, T2Splitted = new String[3];
        String T1Milli, T1Micro, T1Nano = new String();
        String T2Milli, T2Micro, T2Nano = new String();

        T1Splitted = T1.split(":");
        T2Splitted = T2.split(":");

        T1Milli = T1Splitted[0];
        T1Micro = T1Splitted[1];
        T1Nano = T1Splitted[2];

        T2Milli = T2Splitted[0];
        T2Micro = T2Splitted[1];
        T2Nano = T2Splitted[2];

        float Milli, Micro, Nano, distance = 0;

        Milli = Float.valueOf(T2Milli).floatValue()
                - Float.valueOf(T1Milli).floatValue();
        Micro = Float.valueOf(T2Micro).floatValue()
                - Float.valueOf(T1Micro).floatValue();
        Nano = Float.valueOf(T2Nano).floatValue()
                - Float.valueOf(T1Nano).floatValue();

        System.out.println("Milli: " + Milli);
        System.out.println("Micro: " + Micro);
        System.out.println("Nano: " + Nano);

        if(Milli == 0){
            if(Micro == 0){
                if(Nano < 300){
                    distance = Nano * VOP;
                }
                else
                    System.out.println(Nano + " nanoseconds would lead "
                            + "to an unrealistic distance");
            }
            else
                System.out.println(Micro + " microseconds would lead "
                        + "to an unrealistic distance");
        }
        else
            System.out.println(Milli + " microsecods would lead to an "
                    + "unrealistic distance");

        return distance;
    }




//*****************************************************************
    //* Name: ExecQry
*
    //* Input: SQL Query as a String
*
    //* Output: String Array containing the ReseultSet Data
*
    //* Functionality: Returns the ResultSet Data in form of a Multi-
*
    //*               Dimensional Array (according to the RS)
*

//*****************************************************************
    public String[][] ExecQry(String input){

        Connection conn = null;
        String url = "jdbc:mysql://localhost/mydb";
        String user = "root";
        String pass = "malte";
        String strSQL = input;
        ResultSet rs = null;
        String[][] stringResult = null;
        int NoRows = 0;
        int NoColumns = 0;
        int ArrayCount = 0;
        int RSCount = 1;

        try {
                    // Setting the driver and creating a connection
                Class.forName("com.mysql.jdbc.Driver").newInstance();
                conn = DriverManager.getConnection(url, user, pass);
                System.out.println("PosCalcThread: ExecQry: "
                        + "Connected to DB");
                Statement stmt = conn.createStatement();
                rs = stmt.executeQuery(strSQL);

                    //Getting Number of Rows
                rs.last();
                NoRows = rs.getRow();
                rs.first();

                    //Getting Number of Columns
                NoColumns = rs.getMetaData().getColumnCount();

                    //Initializing the string-Array
                stringResult = new String[NoRows][NoColumns];

                    //Writing ResultSet Data in string-Array
                for(int i = 0; i < NoRows; i++){
                    for(int j = 0; j < NoColumns; j++){
                        stringResult[i][j] = rs.getString(j+1);
                    }
                    rs.next();
                }

            }
            catch (Exception e) {
                System.err.println("PosCalcThread: ExecQry: "
                        + "Connection poblem:");
                e.printStackTrace();
            }
            finally {
                if (conn != null) {
                    try {
                        conn.close();
                        System.out.println("PosCalcThread: ExecQry: "
                                + "Connection closed.");
                    } catch(Exception e) {
                        System.out.println("PosCalcThread: ExecQry: "
                                + "Failure while closing connection:");
                        e.printStackTrace();
                    }
                }
            }

        return stringResult;
    }

}
```

Appendix C

Java Code: Graphical User Interface

The following documents contain the Java code of the graphical user interface which is an own program that can be executed by any computer in the network that is able to connect to the localisation database.

C.1 GUI

The GUI class simply creates a new SWING JFrame and adds the ShowLocalisation class, that works as a JPanel to it. ShowLocalisation gets the needed data from the database and paints the objects to its JPanel.

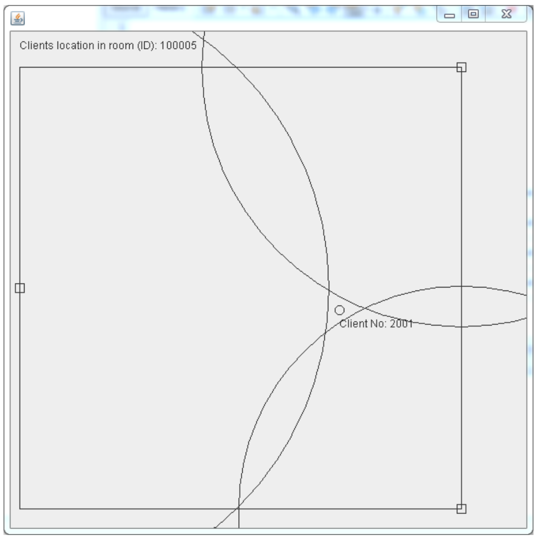


Figure C.1: The Graphical User Interface

C:/Users/Malte Schmidt/Documents/NetBeansProjects/GUI/src/gui/GUI.java

```java

package gui;

import javax.swing.JFrame;

public class GUI {

    public static void main(String[] args) {

        JFrame f = new JFrame();
        f.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
        f.setSize( 600, 600 );
        f.setVisible( true );
        f.add(new ShowLocation());

    }
}
```

1.1 of 1 2012.05.21 17:20:16

C.2 ShowLocation

The ShowLocalisation class extends JPanel and paints the needed objects like the room, the base stations, the distance radius around the base stations and the client to the panel. Therefore it needs to get the according information from the localization database and uses the same classes that also has been used for the calculation of the clients position.

```java

package gui;

import java.awt.*;
import java.sql.*;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.swing.*;
import org.apache.commons.math3.analysis.function.Divide;


public class ShowLocation extends JPanel{



    public void paintComponent(Graphics g){

        super.paintComponents(g);
        Graphics2D g2d = (Graphics2D) g;

        String strSQL;
        String[][] rs;
        int mobileClientID, RoomID, BS1ID, BS2ID, BS3ID;
        double BS1x, BS1y, BS2x, BS2y, BS3x, BS3y;
        double[] coordinates;
        double clientPosX, clientPosY;
        int accuracy = 2;
        int factor = 50;
        int floorID;
        float RoomSizeX, RoomSizeY;
        int localisationID;
        String Date, Time;
        String[] baseStationIDs;
        double RefY = 10;
        int OriginX = 10, OriginY = 40;
        int upperLeftX, upperLeftY, upperRightX, upperRightY;
        int lowerLeftX, lowerLeftY, lowerRightX, lowerRightY;


        strSQL = "SELECT COUNT(idtblposition) "
                + "FROM tblposition";
        rs = ExecQry(strSQL);

        if(Integer.parseInt(rs[0][0]) > 0)
        {
```

```


        strSQL = "SELECT * FROM tblposition "
                + "WHERE Date = (SELECT MAX(Date) FROM tblposition) "
                + "AND Time = (SELECT MAX(Time) FROM tblposition)";
        rs = ExecQry(strSQL);

        mobileClientID = Integer.parseInt(rs[0][1]);
        Date = rs[0][2];
        Time = rs[0][3];
        RoomID = Integer.parseInt(rs[0][4]);
        clientPosX = Double.valueOf(rs[0][5]) * factor;
        clientPosY = (RefY - Double.valueOf(rs[0][6])) * factor;
        System.out.println("Client Position: X(" + clientPosX
                + "), Y(" + clientPosY + ")");
        localisationID = Integer.parseInt(rs[0][7]);

        g2d.drawString("Clients location in room (ID): " + RoomID, 10,
20);


        strSQL = "SELECT * FROM tblroom "
                + "WHERE idtblroom = " + RoomID;
        rs = ExecQry(strSQL);

        floorID = Integer.parseInt(rs[0][2]);
        RoomSizeX = Float.valueOf(rs[0][3]) * factor;
        RoomSizeY = Float.valueOf(rs[0][4]) * factor;
        System.out.println("Room Size: " + rs[0][3] + " * " + rs[0][4]);
        System.out.println("Room Size factored: " + RoomSizeX
                            + " * " + RoomSizeY);

        baseStationIDs = GetInvolvedBaseStations(localisationID);
        BS1ID = Integer.parseInt(baseStationIDs[0]);
        BS2ID = Integer.parseInt(baseStationIDs[1]);
        BS3ID = Integer.parseInt(baseStationIDs[2]);

        coordinates = GetBSCoordinates(BS1ID);
        BS1x = coordinates[0] * factor;
        BS1y = (RefY - coordinates[1]) * factor;

        coordinates = GetBSCoordinates(BS2ID);
        BS2x = coordinates[0] * factor;
        BS2y = (RefY - coordinates[1]) * factor;

```

```java
        coordinates = GetBSCoordinates(BS3ID);
        BS3x = coordinates[0] * factor;
        BS3y = (RefY - coordinates[1]) * factor;



            // Calculate the points according to the size of the room
        upperLeftX = OriginX;
        upperLeftY = OriginY;

        upperRightX = (int) (OriginX + RoomSizeX);
        upperRightY = OriginY;

        lowerLeftX = OriginX;
        lowerLeftY = (int) (OriginY + RoomSizeY);

        lowerRightX = (int) (OriginX + RoomSizeX);
        lowerRightY = (int) (OriginY + RoomSizeY);

            // Linien Zeichnen
        g2d.drawLine( upperLeftX, upperLeftY, upperRightX, upperRightY );
        g2d.drawLine( upperLeftX, upperLeftY, lowerLeftX, lowerLeftY );
        g2d.drawLine( lowerLeftX, lowerLeftY, lowerRightX, lowerRightY );
        g2d.drawLine( upperRightX, upperRightY, lowerRightX, lowerRightY);


        g2d.drawOval((int)clientPosX - 5 + OriginX,
                (int)clientPosY - 5 + OriginY, 10, 10);
        g2d.drawString("Client No: " + mobileClientID,
                (int)clientPosX + OriginX,
                (int)clientPosY + 20 + OriginY);

        g2d.drawRect((int)BS1x - 5 + OriginX,
                (int)BS1y - 5 + OriginY, 10, 10);

        g2d.drawRect((int)BS2x - 5 + OriginX,
                (int)BS2y - 5 + OriginY, 10, 10);

        g2d.drawRect((int)BS3x - 5 + OriginX,
                (int)BS3y - 5 + OriginY, 10, 10);



        String[] TimestampsBS1, TimestampsBS2, TimestampsBS3;
        float RadiusBS1, RadiusBS2, RadiusBS3;

             // Get the timestamps of BS1
        TimestampsBS1 = GetTimestamps(localisationID, BS1ID);

            // Get the radius of BS1
        RadiusBS1 = TimestampsToDistance(TimestampsBS1);
        System.out.println("RBS1: " + RadiusBS1);
        RadiusBS1 = RadiusBS1 * factor * 2;
        System.out.println("RBS1 factored: " + RadiusBS1);

            // Get the timestamps of BS2
        TimestampsBS2 = GetTimestamps(localisationID, BS2ID);

            // Get the radius of BS2
        RadiusBS2 = TimestampsToDistance(TimestampsBS2);
        System.out.println("RBS2: " + RadiusBS2);
        RadiusBS2 = RadiusBS2 * factor * 2;
        System.out.println("RBS2 factored: " + RadiusBS2);


            // Get the timestamps of BS3
        TimestampsBS3 = GetTimestamps(localisationID, BS3ID);

            // Get the radius of BS3
        RadiusBS3 = TimestampsToDistance(TimestampsBS3);
        System.out.println("RBS3: " + RadiusBS3);
        RadiusBS3 = RadiusBS3 * factor * 2;
        System.out.println("RBS3 factored: " + RadiusBS3);

        Divide d = new Divide();


        g2d.drawOval((int)BS1x - (int)d.value(RadiusBS1, 2) + OriginX,
                (int)BS1y - (int)d.value(RadiusBS1, 2) + OriginY,
                (int)RadiusBS1,
                (int)RadiusBS1);

        g2d.drawOval((int)BS2x - (int)d.value(RadiusBS2, 2) + OriginX,
                (int)BS2y - (int)d.value(RadiusBS2, 2) + OriginY,
                (int)RadiusBS2,
                (int)RadiusBS2);

        g2d.drawOval((int)BS3x - (int)d.value(RadiusBS3, 2) + OriginX,
                (int)BS3y - (int)d.value(RadiusBS3, 2) + OriginY,
                (int)RadiusBS3,
                (int)RadiusBS3);

        }


    }




//*****************************************************************
    //* Name: ExecQry
*
    //* Input: SQL Query as a String
*
    //* Output: String Array containing the ReseultSet Data
*
    //* Functionality: Returns the ResultSet Data in form of a Multi-
*
    //*                Dimensional Array (according to the RS)
*

//*****************************************************************
    public String[][] ExecQry(String input){

        Connection conn = null;
        String url = "jdbc:mysql://localhost/mydb";
        String user = "root";
        String pass = "malte";
        String strSQL = input;
        ResultSet rs = null;
        String[][] stringResult = null;
        int NoRows = 0;
        int NoColumns = 0;
        int ArrayCount = 0;
        int RSCount = 1;

        try {
                Class.forName("com.mysql.jdbc.Driver").newInstance();
                conn = DriverManager.getConnection(url, user, pass);
                System.out.println("PosCalcThread: ExecQry: "
                        + "Connected to DB");
                Statement stmt = conn.createStatement();
                rs = stmt.executeQuery(strSQL);

                    //Getting Number of Rows
                rs.last();
                NoRows = rs.getRow();
                rs.first();

                    //Getting Number of Columns
                NoColumns = rs.getMetaData().getColumnCount();

                    //Initializing the string-Array
                stringResult = new String[NoRows][NoColumns];

                    //Writing ResultSet Data in string-Array
                for(int i = 0; i < NoRows; i++){
                    for(int j = 0; j < NoColumns; j++){
                        stringResult[i][j] = rs.getString(j+1);
                    }
                    rs.next();
                }

            }
            catch (Exception e) {
                System.err.println("PosCalcThread: ExecQry: "
                        + "Connection poblem:");
                e.printStackTrace();
            }
            finally {
                if (conn != null) {
                    try {
                        conn.close();
                        System.out.println("PosCalcThread: ExecQry: "
                                + "Connection closed.");
                    } catch(Exception e) {
                        System.out.println("PosCalcThread: ExecQry: "
                                + "Failure while closing connection:");
                        e.printStackTrace();
                    }
                }
            }

        return stringResult;
    }

```

C:/Users/Malte Schmidt/Documents/NetBeansProjects/GUI/src/gui/ShowLocation.java

```

//*******************************************************************
    //* Name: GetBSCoordinates
*
    //* Input: Base Station ID as String
*
    //* Output: X-Y-Coordinates of the BS as a float Array
*
    //* Description: Gets the X-Y-Coordinates of the Base Station from
*
    //*             the according table of the DB
*
    //* float coordinates[0] = x-coordinate
*
    //* float coordinates[1] = y-coordinate
*
//*******************************************************************
    public double[] GetBSCoordinates(int BaseStationID){

        int BSID = BaseStationID;

        String[][] rs;
        int count = 0;
        int NoRows = 0;
        int NoColumns = 0;
        double[] coordinates = new double[2];

        String strSQL = "SELECT RoomRelPosX, RoomRelPosY "
                + "FROM tblbasestation "
                + "WHERE idtblbasestation = " + BSID;
        rs = ExecQry(strSQL);

            //Getting number of rows and columns
        NoRows = rs.length;
        NoColumns = rs[0].length;

            //Getting Base Station IDs from ResultSet string-Array
        for(int i = 0; i < NoRows; i++){
            for(int j = 0; j < NoColumns; j++){
                coordinates[j] = Double.parseDouble(rs[i][j]);
            }
        }

```

7.1 of 11 2012.05.22 12:49:03

```
        return coordinates;
    }



//*******************************************************************
    //* Name: GetInvolvedBaseStations
*
    //* Input: Localisation ID as Integer
*
    //* Output: Localisation Involved Base Stations as a String Array
*
    //* Description: Gets the Base Stations involved in the
Localisation*
    //*              according to the localisation ID from the DB
*

//*******************************************************************
    public String[] GetInvolvedBaseStations(int LocalID){


        String[] BaseStations = new String[3];
        String[][] rs;
        int count = 0;
        int NoRows = 0;
        int NoColumns = 0;

        String strSQL = "SELECT BaseStationID "
                + "FROM tblframe WHERE "
                + "LocalisationID = " + LocalID;
        rs = ExecQry(strSQL);

            //Getting number of rows and columns
        NoRows = rs.length;
        NoColumns = rs[0].length;

            //Getting Base Station IDs from ResultSet string-Array
        for(int i = 0; i < NoRows; i++){
            for(int j = 0; j < NoColumns; j++){
                BaseStations[i] = rs[i][j];
            }
        }

        System.out.println("Involved Base Stations: " + BaseStations[0]
                + ", " + BaseStations[1] + ", " + BaseStations[2]);

        return BaseStations;
    }




//*******************************************************************
    //* Name: TimestampToDistance
*
    //* Input: Two Timestamps recorded at one BS as a String Array
*
    //* Output: The distance in meters between the BS and the MC
*
    //* Functionality: Calculates the distance according to the
*
    //*                Timestamps
*
    //*
*

//*******************************************************************
    public float TimestampsToDistance(String[] Timestamps) {

        float VOP = (float) 0.28; // Velocity of Propagatio
                                  // (~28cm/nanosecond)
        String T1 = Timestamps[0];
        String T2 = Timestamps[1];
        String[] T1Splitted, T2Splitted = new String[3];
        String T1Milli, T1Micro, T1Nano = new String();
        String T2Milli, T2Micro, T2Nano = new String();

        T1Splitted = T1.split(":");
        T2Splitted = T2.split(":");

        T1Milli = T1Splitted[0];
        T1Micro = T1Splitted[1];
        T1Nano = T1Splitted[2];

        T2Milli = T2Splitted[0];
        T2Micro = T2Splitted[1];
        T2Nano = T2Splitted[2];

        float Milli = 0, Micro = 0, Nano = 0, distance = 0;

```

```java
        Milli = Float.valueOf(T2Milli).floatValue()
                - Float.valueOf(T1Milli).floatValue();
        Micro = Float.valueOf(T2Micro).floatValue()
                - Float.valueOf(T1Micro).floatValue();
        Nano = Float.valueOf(T2Nano).floatValue()
                - Float.valueOf(T1Nano).floatValue();

        System.out.println("Milli: " + Milli);
        System.out.println("Micro: " + Micro);
        System.out.println("Nano: " + Nano);

        if(Milli == 0){
            if(Micro == 0){
                if(Nano < 300){
                    distance = Nano * VOP;
                }
                else
                    System.out.println(Nano + " nanoseconds would "
                            + "lead to an unrealistic distance");
            }
            else
                System.out.println(Micro + " microseconds would "
                        + "lead to an unrealistic distance");
        }
        else
            System.out.println(Milli + " microsecods would "
                    + "lead to an unrealistic distance");

        System.out.println("Distance for TS " + Timestamps[0] + " and "
                + Timestamps[1] + " is " + distance);

        return distance;
    }



//*******************************************************************
    //* Name: GetTimestamps
*
    //* Input: Localisation ID and BaseStation ID as String Array
*
    //* Output: Timestamps according to LocID and BS ID
*
    //* Description: Gets the Timestamps of one measurement for one
*
```

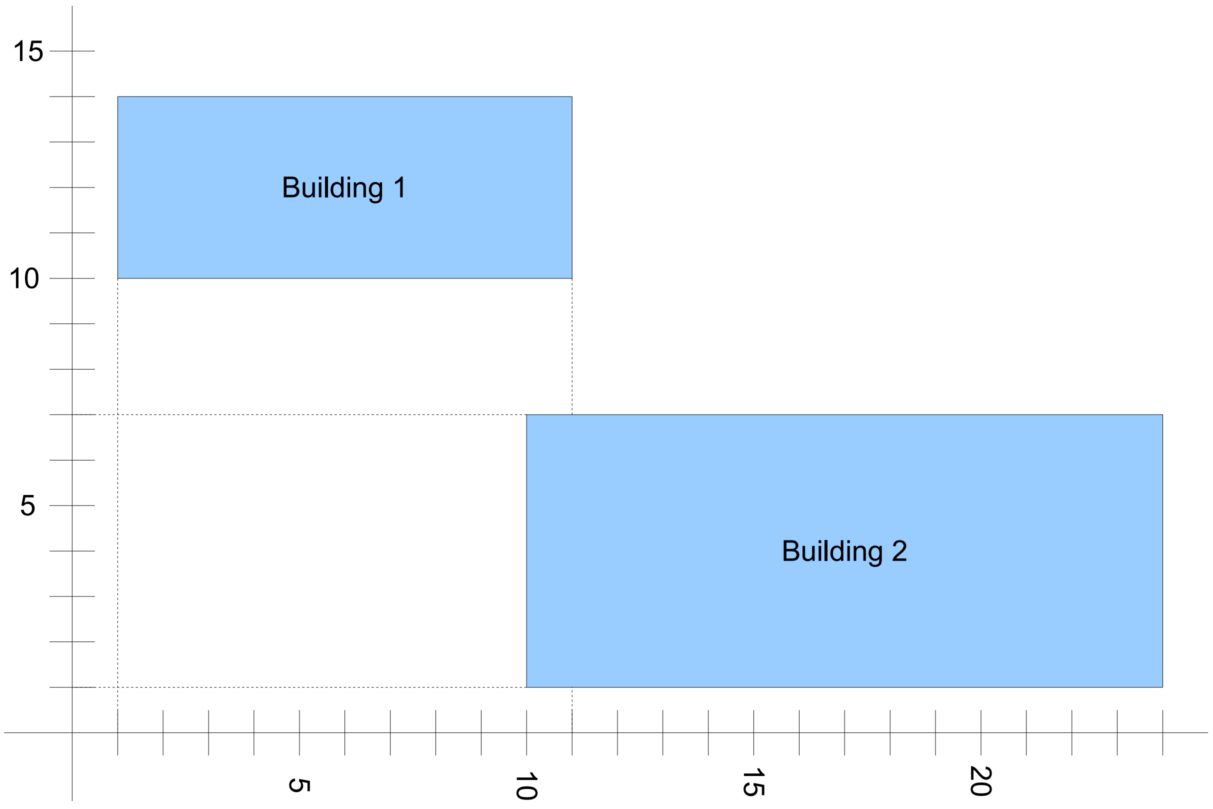
```java
    //*              Localisation ID and one Base Station from the DB *
//*******************************************************************
    public String[] GetTimestamps(int localisationID, int BaseStationID){

        String[] output = new String[2];
        int locID = localisationID;
        int BSID = BaseStationID;
        int NoRows, NoColumns;
        String[][] rs;
        String strSQL = "SELECT OutgoingTime, IncomingTime"
                + " FROM tblframe"
                + " WHERE LocalisationID = " + locID
                + " AND BaseStationID = " + BSID;
        rs = ExecQry(strSQL);

            //Getting number of rows and columns
        NoRows = rs.length;
        NoColumns = rs[0].length;

            // Getting Base Station IDs from ResultSet string-Array
            // NOTE: Check to replace with System.arraycopy
        for(int i = 0; i < NoRows; i++){
            for(int j = 0; j < NoColumns; j++){
                output[j] = rs[i][j];
            }
        }

        System.out.println("TS for BSID: " + BSID + " is: "
                + output[0] + ", " + output[1]);

        return output;
    }

}
```

Appendix D

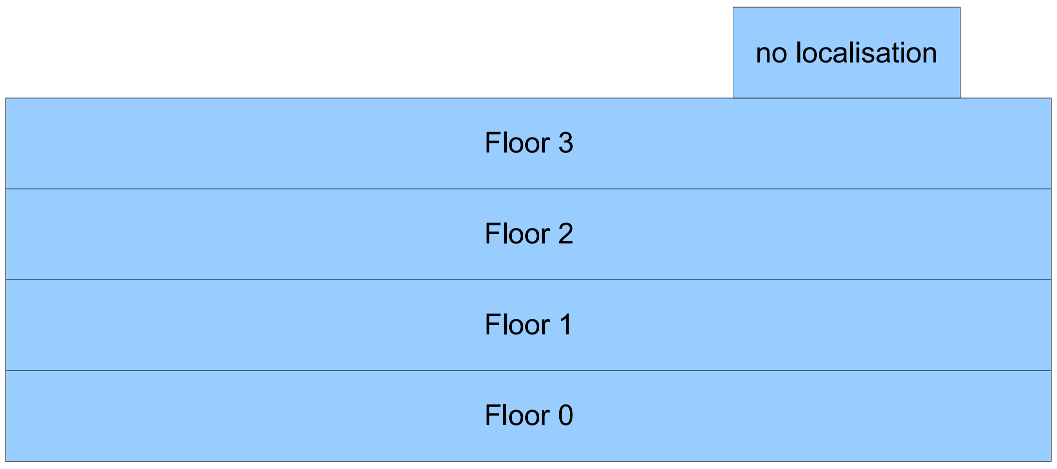
Localisation Area Test Case

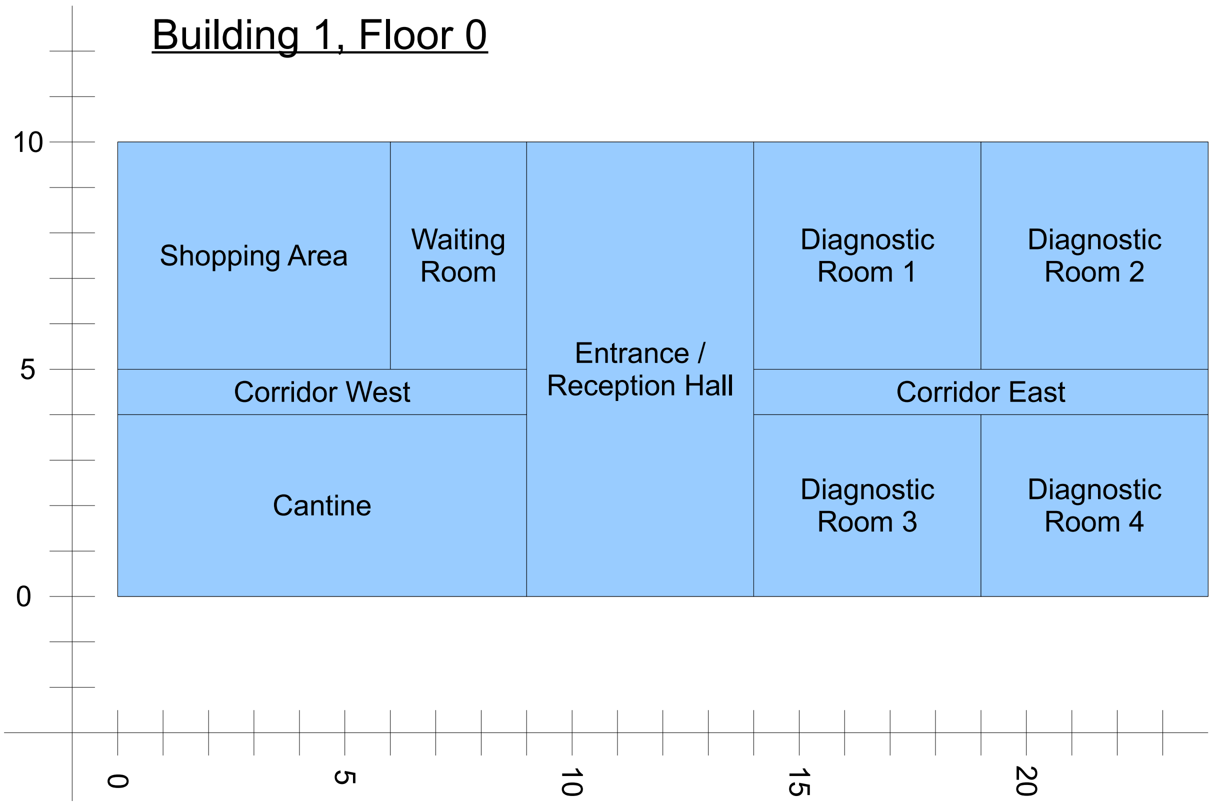
The localisation area test case contains of the definition of a localisation area for proof the developed algorithm. The first page depicts two different buildings at a area, the second page the different floors of the area. Page number three is a detailed overview of the first floor of building one including all rooms and corridors. The fourth and last page is a detailed view of one room including the installed base stations and their positions in relation to the room.

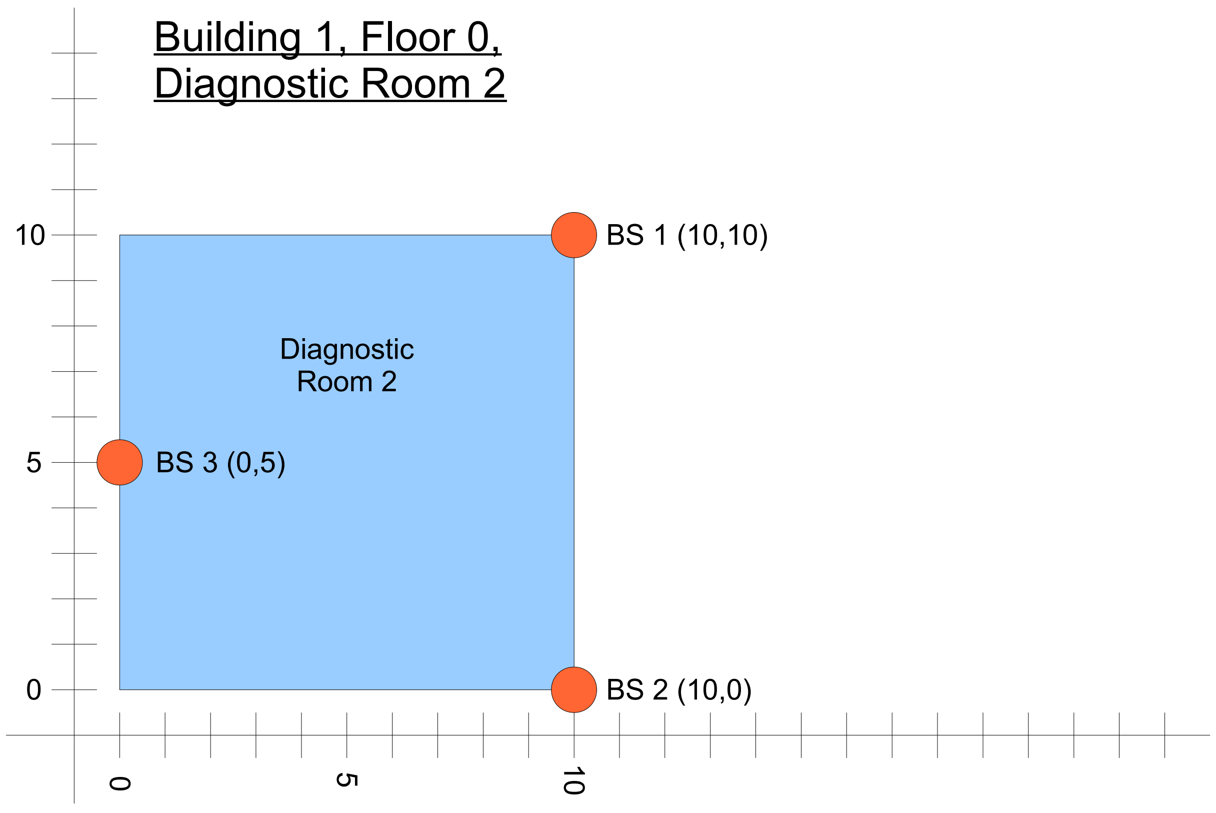
All pages use an own scale and the base stations at the last page use room related coordinates. This is a very easy way of defining the mobile clients position and allows a easy measuring of the base stations installation points, especially when no detailed floor layouts are available.


15
10
5
Building 1
Building 2
5
10
15
20

Building 1




Building 1, Floor 0
Shopping Area
Waiting Room
Entrance / Reception Hall
Diagnostic Room 1
Diagnostic Room 2
Corridor West
Corridor East
Cantine
Diagnostic Room 3
Diagnostic Room 4
10
5
0
0
5
10
15
20


Building 1, Floor 0,
Diagnostic Room 2
Diagnostic
Room 2
BS 1 (10,10)
BS 2 (10,0)
BS 3 (0,5)
10
5
0
0
5
10