ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Automatic Control Laboratory

Scalable Ultra-wideband localisation

with application to autonomous video production

Group Project

Yvan Bosshard **Tiago Salzmann**

21. October 2018

Abstract

Recent development in indoor localisation involving Ultra-wideband (UWB) technology suggests its potential as a cheap, robust and scalable localisation system. The Automatic Control Lab (IfA) has supervised multiple student projects involving UWB technology: one concluded by a participation in the Microsoft Indoor Localisation Challenge and another enabled UWB-localisation for the Crazyflie quad-rotors used in the P&S course at IfA. The accuracy of the systems resulting from these projects varied between 10 and 50 centimeters, however neither system is suitable for use in arbitrary environments and with a varying number of moving agents (tags) to be localised.
The motivating application for this project is autonomous video production, where the number and location of anchors needs to adapt to the environment, and the number of moving objects to be tracked and filmed varies. The goal of this group project is to take the experiences from the previous UWB projects and develop a system that can be used in an arbitrary environment and natively supports a varying number of anchors and tags.

Preface

The importance of indoor localisation increases with the ever rising number of autonomous systems performing a variety of tasks, especially where current localisation technology can not be used due to environmental restrictions or the accuracy they provide. In contrast to the Global Positioning System (GPS), which is strongly affected by the environment as a result of the technology, an indoor system can provide localisation for buildings, tunnels or mines. On the other hand, the position accuracy of the GPS is specified to achieve an error of less than 5 m [1], which is not suitable for small agents or in situations where a greater precision is required (e.g. quad-copters, autonomous video production).

The approach taken to provide localisation in these scenarios is a Time Difference of Arrival (TDoA) scheme, as the number of devices that can be localised simultaneously is not limited, similar to GPS, where the tag is only listening to messages from the anchors. The resulting system should be easy to use, fast to setup and provide a decent position update rate.

Contents

List of Figures

Nomenclature

c	Speed of Light in vacuum
$d_{i,j}$	Distance between devices i and j
t	Timestamp on a reference time scale
$t^{(i)}$	Timestamp measured on device i
$\tilde{t}^{(i)}$	Sending timestamp on device i
$\hat{t}_{i,j}$	Time of flight between devices i and j; $\hat{t}_{i,j} = d_{i,j}/c$
τ_i	Time Difference of Arrival of the message from anchor i with respect to the reference anchor
ω_i	Distance difference between the distances to anchor i and the reference anchor; $\omega_i = \tau_i \cdot c$
$\mathbf{\Gamma}_i$	Data vector of anchor i, used in the localisation process
Δ_i	Clock offset of clock i
κ_i	Clock drift of clock i

Acronyms

GPS Global Positioning System. 5, 13, 14, 21, 23, 25, 40

GUI Graphical User Interface. 36

TDoA Time Difference of Arrival. 5, 13, 15–19, 21, 25, 28–30, 33, 36, 38–40

TWR Two-way-ranging. 13, 21

UWB Ultra-wideband. 3, 13, 14, 33–35, 39

Chapter 1

The Idea

Previous work has proven UWB as a feasible technology for precise indoor localisation, allowing for centimeter accuracy [2]. Using Two-way-ranging (TWR), a process consisting of sending three messages back and forth between an anchor and a tag being localised, the time of flight can be measured and the distance between them calculated. By repeating this process for at least two other anchors, a unique position in 3D-space can be estimated, assuming the anchor positions are known. This however comes at a few drawbacks: Not only is TWR a rather slow process, the tag also has to be active (i.e. send messages) in order to be able to localise itself. If multiple tags need to be localised, each one of them must send messages to all of the anchors and some sort of coordination has to be incorporated to prevent multiple messages being sent at the same time. [2] enabled TWR-based localisation on a single nano-quadcopter used at the P&S course at Ifa with an achieved positioning frequency that is barely usable. Increasing the number of tags decreases the rate further, making the scheme unsuitable for more than one device.

To solve the issues introduced, a GPS-like approach was chosen for this project using TDoA measurements. The mathematical theory is explained in detail in section 3. The project aimed to create a system to provide localisation for different scenarios found in autonomous video production. The requirements for such a system included a fast and convenient setup, low-power devices and fast localisation rate.

1.1 Advantages

The most important advantages of a TDoA scheme are summarised below:

Passive tags TDoA measurements can be achieved without a tag having to actively send any messages. Like a GPS receiver, it only listens to data being broadcasted by the anchors (satellites). This not only decreases power consumption, allowing the system to be incorporated into low-power applications, but also eliminates the need for coordination between the tags.

Arbitrary number of tags Without the need for coordination between the tags, a running system can be extended by any number of tags, most importantly without reducing the positioning rate.

Low power As the tag is passive, power intensive sending of messages can be avoided.

Small devices With a new generation of UWB devices, the whole system can be run with comparatively small devices, that can be integrated into existing hardware, using a serial connection.

1.2 Challenges

The aforementioned benefits and the requirements set for the system come at some crucial challenges, strongly affecting performance of the resulting system.

Clock synchronisation While GPS satellites carry atomic clocks, the small devices used need another way to reliably time their actions. Badly synchronised clocks inherently affect the precision of the localisation.

Fast localisation Albeit not being affected by multiple tags, the rate of localisation still is limited by numerous factors, e.g. propagation time of the signals, time to process a message or to calculate location.

Ease of use To allow the system to be easy to use, easy to maintain, fast to set up and convenient to adapt, a good amount of time needs to be dedicated to a carefully structured architecture.

Solutions to these challenges, trying to make full use of the advantages, are explained in detail in this report.

Chapter 2

System Architecture

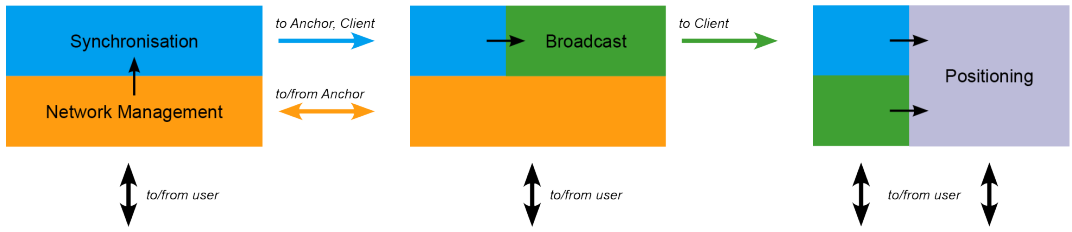
The architecture of the system is partitioned into four distinct parts, each of which enables a certain set of features, described below. Figure 2.1 visualises the four components **Network Management**, **Synchronisation**, **Broadcast** and **Positioning**, and their connections among each other. The different services run in different combinations and to various extent, depending on the role of a device. Inside a network, there are three roles a device can take. The **Master** is a node unique in every network, responsible for setting up and managing the network. It defines parameters, registers new anchors and allows other devices to synchronise their clocks. It monitors the status of devices and is also needed to start and stop broadcasts so a client can determine its position. These broadcast messages come from **Anchors**, whose locations are known, similar to satellites in the GPS. Their clocks are synchronised by the master and they can provide the master with status and health information about themselves. The **Client** is the actual tag, that is being localised. It receives clock synchronisation and broadcast messages from the master and anchors respectively to use in the positioning service to calculate a position. This position, amongst other information, can be retrieved by the user via an interface.

2.1 Network Management

To allow the system to be easy to set up and easy to maintain, the Network Management service was designed with a high level of abstraction and, where possible, with automation in mind. The main part of this service is running on the master device, as it acts as a control interface for the whole network. It's most important features are explained below.

2.1.1 Anchor Registration

In order for a client to be able to calculate its position, at least four anchors need to be present and working in the system. When an anchor is initialised, it request a unique network address, similar to an IP address in a computer network, which is provided by the master. It keeps track of all anchors in the network with their assigned addresses, positions and status. Before an anchor can participate in the broadcasting schedule, it needs to be assigned a position. This can either happen manually by entering the respective coordinates of the device, or as part of the initialisation process, where all the



(a) Master schematic (b) Anchor schematic (c) Client schematic

Figure 2.1: Schematic view of the system architecture and their connections. While **Network Management** serves as a mean to administer and interface with the network, **Synchronisation** is used to accurately synchronise the individual clocks. Finally **Broadcast** contains all necessary information that allow the **Positioning** service running on a tag to compute its position in 3D space. Synchronisation messages from a master node reach all devices, while broadcasts are only considered by client nodes. Administrative messages are exchanged between the master and multiple anchor nodes. Every device features a user interface that allows the access to information of the respective service.

anchors perform a localisation using trilateration, to determine all their positions. Once a position is assigned, an anchor is ready to broadcast it's position as part of the network.

2.1.2 System Configuration

The requirements in terms of localisation speed vary widely depending on the use scenario. While a robot in a factory might move with a few kilometers per hour, a race car on a track can reach a multiple of that. For the former, a few position updates per second might suffice, since it will move only some meters, whereas the latter will move dozens. To allow the system to be used in any of theses scenarios, the positioning rate, amongst other related parameter is adjustable from the master interface. From arbitrarily slow [1] up to 87 Hz, the frequency can be arbitrarily chosen, keeping some limitations (see also sec. 2.3) in mind.

2.1.3 System Monitoring

Since the system can be used in arbitrary environments, the anchors don't need to be connected to a host device (computer). Nonetheless, to allow the administrator of the network to get information about the status of the network and the individual devices, anchors are capable to transmit their status to the master node. A user can request status reports from individual anchors, containing vital information such as supply voltage, chip temperature, assigned network address and the devices location. To reduce user effort, these details are repeatedly checked autonomously by the master node, collecting status information about the entire network and notifying the user about any anomalies, such as low battery or overheated chip of an anchor. In case of such a situation, the corresponding anchor is automatically excluded from the broadcasting schedule as not to damage the hardware or drain the battery even more. Since all the status information is preserved, after fixing the issue, the device can be integrated into the network readily.

[1] Actually the lowest possible update frequency is limited by the data type used to store the localisation period, which would correspond to 233 pHz.

2.2 Synchronisation

The accuracy of TDoA measurements are heavily affected by the precision of the clocks used in the devices. Where the GPS copes with this challenge by using atomic clocks on each device (satellite), a system offering comparable precision in this setup would not be suitable in terms of size and power consumption. To synchronise two clocks, two quantities need to be taken into account: Clock offset and clock drift.

2.2.1 Clock Offset

Clock offset is the difference between the values two clocks tell at the same moment in time. Imagine having two wall clocks A and B, A telling 2 p.m., B telling 3:15 p.m., when your wrist watch actually tells 2:45 p.m. Taking the wrist watch as a reference, the offset of clock A would then be -45 minutes and the offset of clock B would be 30 minutes. More formally, the offset of a clock i can be defined as

$$\Delta_i = t^{(i)} - t^{(r)}, \tag{2.1}$$

where $t^{(i,r)}$ are the times on clock i and the reference clock r respectively at the same moment in time. This definition implies, that a clock with a positive offset runs ahead, one with a negative offset runs behind a reference clock. The reasons for clock offset are diverse, but in this setup the two most important are: Devices are switched on (their clock started) at different times, and their clocks do not run at the same rate. Whatever the reason, the way it is handled stays the same. In this system, the master clock is used as a reference. Although this clock cannot achieve the same precision as for instance an atomic clock might deliver, it is accurate enough to allow for localisation with decent precision. In order to synchronise two clocks in terms of offset, the following approach was chosen:

1. Determine timestamp of synchronisation $t_{SI}^{(r)}$ on the reference clock[2].
2. Send synchronisation message at $t_{SI}^{(r)}$.
3. Note timestamp of reception on the remote device $t_{SI}^{(i)}$.
4. Using the known distance between the devices $d_{r,i}$, calculate the time of flight of the message $\hat{t}_{r,i}$.
5. Correct eq. 2.1 by the time of flight:

$$\Delta_i = t_{SI}^{(i)} - (t_{SI}^{(r)} + \hat{t}_{r,i}). \tag{2.2}$$

This procedure assumes, that the clocks run at the same rate, at least for the duration of $\hat{t}_{r,i}$. Since the propagation time is comparably small to the synchronisation interval, this assumption is valid. Also noteworthy is the fact, that since the offset calculation uses the known distance between the devices, this cannot be done for a tag, since the master node has no information about any tags present, especially not about their location. It turns

[2]The desired timestamps for synchronisation $t_{SI}^{(r)}$ are network parameters, that are known to all devices connected to the network.

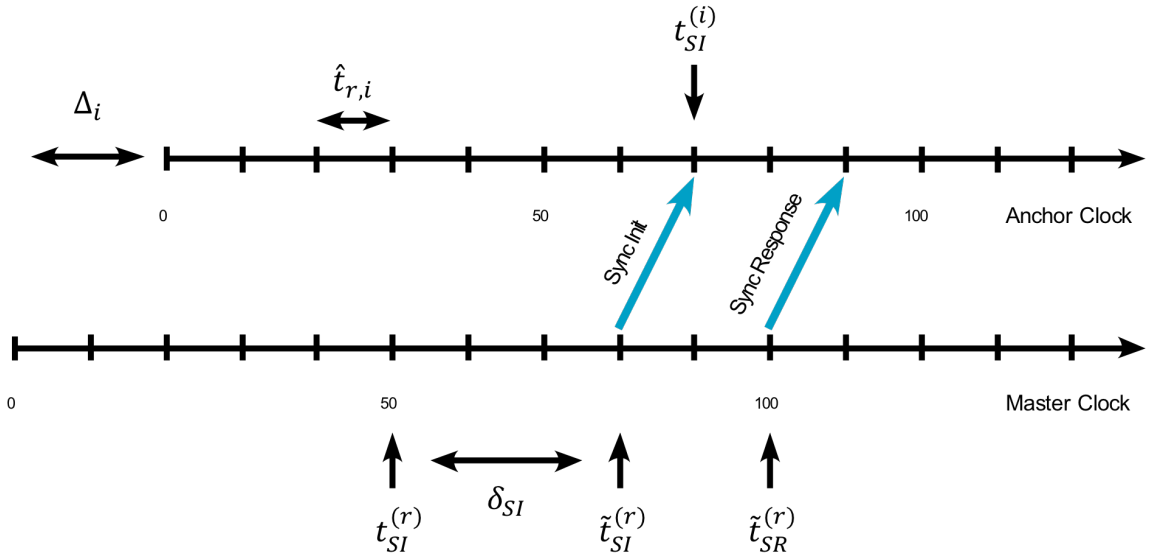


Figure 2.2: Schematic view of the synchronisation process. Since the sending time $t_{SI}^{(r)}$ cannot be chosen arbitrarily, the correction δ_{SI} has to be calculated and provided in a follow-up message SR.

out, offset is not a crucial variable for the localisation of the tag, since the timestamps of arrival of the broadcast messages are all measured in the time frame of the client and only time differences are used.
Furthermore, the hardware used, in fact does allow to specify a timestamp when a message should be sent, but this timestamp cannot be chosen with the same resolution as the clock offers. Instead, the message will be sent within a small interval around the desired time and the actual timestamp can be recovered afterwards. This introduces an uncertainty in eq. 2.2 which only can be corrected by knowing the exact time $(t_{SI}^{(r)} + \delta_{SI})$. This uncertainty can be calculated by comparing the desired sending time with the actual one and can therefore be defined as

$$\delta = \tilde{t} - t, \tag{2.3}$$

where $\tilde{t}$ is the actual sending time. A second message can then be sent containing δ_{SI} for the receiving device to use. Inserting into eq. 2.2 and replacing $t^{(r)}$ with $\tilde{t}^{(r)}$ yields the formula used to calculate the clock offset:

$$\begin{aligned} \Delta_i &= t_{SI}^{(i)} - (\tilde{t}_{SI}^{(r)} + \hat{t}_{r,i}) \\ &= t_{SI}^{(i)} - \left(\left(t_{SI}^{(r)} + \delta_{SI}\right) + \hat{t}_{r,i}\right). \end{aligned} \tag{2.4}$$

A schematic view of the synchronisation process is depicted in figure 2.2.

2.2.2 Clock Drift

Clock drift is the difference in time intervals two clocks tell. Using the analogy above, imagine measuring 15 minutes on the wrist watch. When starting the timing process, clocks A and B both show 2 p.m. After 15 minutes, clock A tells 2:17 p.m. and clock B 2:11 p.m. While the offset was 0 at the beginning, after 15 minutes it is 2 minutes and -4 minutes respectively. This process can be formalised using t_0 and t_1 as the reference (true) timestamps of the start and end of the timing respectively:

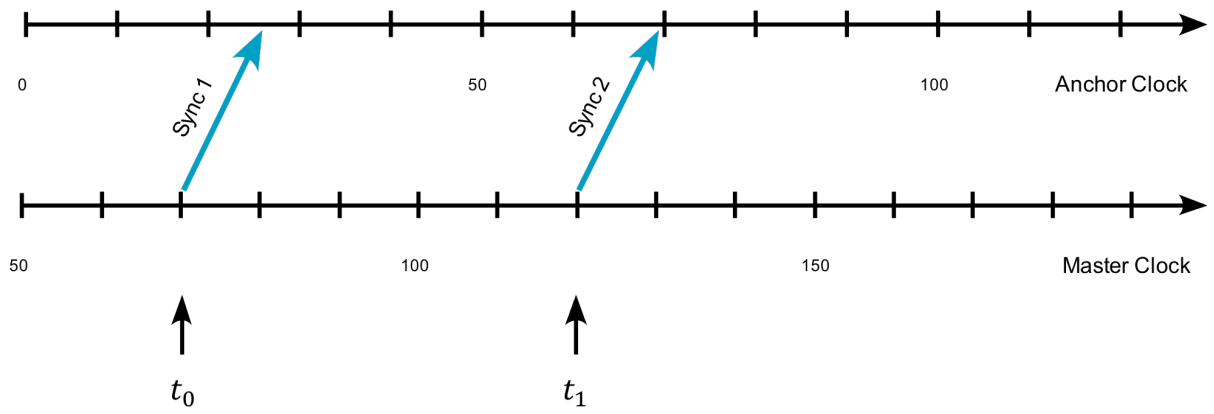


Figure 2.3: Schematic view of clock drift synchronisation. *Sync 1* and *Sync 2* denote two independent synchronisation processes (c.f. figure 2.2). The process allows to determine $\Delta(t_0)$ and $\Delta(t_1)$ in order to calculate the clock drift.

$$\kappa_i = \Delta_i(t_1) - \Delta_i(t_0). \tag{2.5}$$

It should be noted, that since the clocks now don't necessarily run at the same rate, the offset becomes a function of time. A positive offset means, that a clock is running faster than the reference, a negative value analogously means, the clock is running slower than the reference clock. Knowing the clock drift, one can correct for how much a clock counts too much or too little between two offset calculations. Combining clock offset and clock drift, one can translate the timestamps of the remote clock into the reference time frame. This scheme will be used in section 2.3 and a visual interpretation can be found in figure 2.3.

2.3 Broadcast

All explanations in this chapter are made using the assumption, that the clocks of different devices are synchronised by the means detailed in section 2.2. This means that the clocks are corrected for clock offset and clock drift. The implication can be formalised as follows:

$$t = t^{(i)} \quad \forall i. \tag{2.6}$$

2.3.1 Information

In order to allow localisation using a TDoA scheme, the device being localised needs information about an anchor, consisting of the position $\mathbf{r}$ and the TDoA value τ. Since the localisation algorithm uses distance differences ω and not time differences (see also section 3), using the speed of light, they can be converted into one another using

$$\omega_i = c \cdot \tau_i. \tag{2.7}$$

The set of data corresponding to an anchor i is denoted by

$$\mathbf{\Gamma}_i = [\mathbf{r}_i, \quad \omega_i]^T = [x_i, y_i, z_i, \omega_i]^T. \tag{2.8}$$

Before a position can be computed, the vector Γ of at least four anchors needs to be known.

2.3.2 Time Difference of Arrival

The way the TDoA values are computed will be explained by first following an intuitive approach and refining it step wise to avoid the disadvantages introduced. This approach relies on the fact, that all clocks (also the one on the client) are synchronised. At a given point in time t_{BR}, known to all devices on the network, every anchor sends out a message containing its identifier. Due to the difference in distance $d_{i,c} - d_{j,c}$ between a client and anchors i and j respectively, the messages will (most of the time) not arrive at the same time. If the timestamp of arrival of a message from anchor i at the client is denoted with $t^{(c)}_{BR,i}$, then the time of flight (and thus the distance) could easily be calculated using

$$\hat{t}_{i,c} = t^{(c)}_{BR,i} - t_{BR}. \tag{2.9}$$

However, as explained in section 2.2.1 the clocks of clients are not synchronised for clock offset, which prevents the use of equation 2.9, since the value t_{BR} has no meaning in the frame of reference of an unsynchronised clock. Using equation 2.9 nonetheless, the **difference in propagation time** between the messages of two anchors can be calculated as

$$\begin{aligned} \tau_{i,j} &= \hat{t}_{i,c} - \hat{t}_{j,c} \\ &\overset{(2.9)}{=} (t^{(c)}_{BR,i} - t_{BR}) - (t^{(c)}_{BR,j} - t_{BR}) \\ &= t^{(c)}_{BR,i} - t^{(c)}_{BR,j}. \end{aligned} \tag{2.10}$$

An interpretation for difference in distance and arrival time is depicted in figure 2.4. As it turns out, the difference in propagation time is equal to the difference in arrival time.

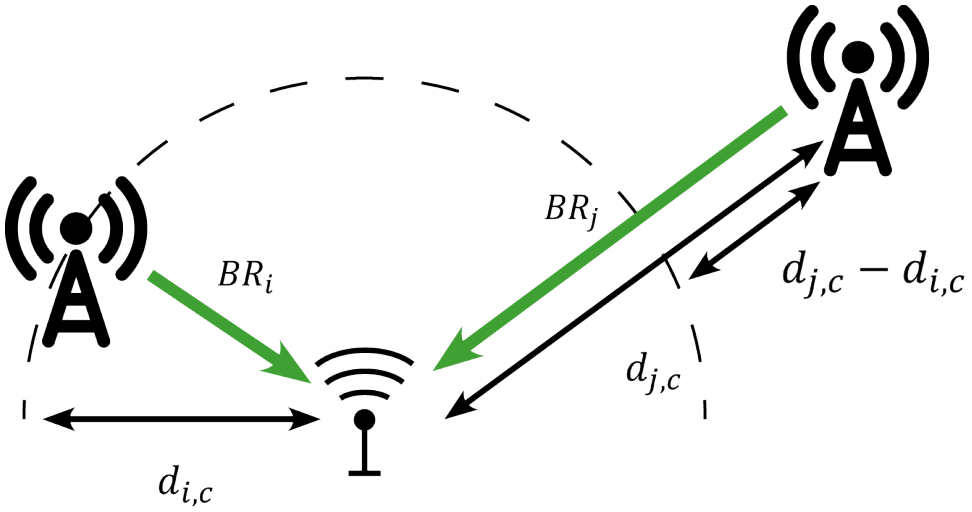


Figure 2.4: Interpretation of the term *Distance Difference*. The signals from anchors i and j were emitted at the same time and travel different distances, resulting in a different time of arrival at the client. Using anchor i as a reference, the TDoA value of anchor j can be expressed as $\tau_j = \omega_j / c = (d_{j,c} - d_{i,c})/c$.

This might seem intuitive, but the fact that this calculation does not rely on the sending time point is crucial as it allows the client to work without offset correction. The only requirements are the synchronisation of the anchors and the clock drift correction on the client.

The equations 2.7 and 2.8 suggest, that the TDoA values τ_i are functions of the corresponding anchors only, but this is not the case. Since time differences are considered, a reference time point needs to be considered. This point t_r could be chosen arbitrarily and the resulting time differences of arrival of an anchor i is obtained by

$$\tau_i = t^{(c)}_{BR,i} - t_r. \tag{2.11}$$

Despite not losing any information, a different transformation that is useful in the localisation problem can be made, by considering the time of arrival of a reference anchor as reference time: $t_r := t^{(c)}_{BR,r}$. Inserting this into equation 2.11 yields the final statement to calculate the TDoA values:

$$\begin{aligned} \tau_i &= t^{(c)}_{BR,i} - t_r \\ &= t^{(c)}_{BR,i} - t^{(c)}_{BR,r}. \end{aligned} \tag{2.12}$$

Per definition, the TDoA value of the reference anchor will be zero $\tau_r \equiv 0$.
The scheme introduced up to this point would work, if messages can be processed as fast as they arrive. Since the manufacturer claims 10 cm precision, this is a desirable distance to be able to resolve. It takes light 0.33 ns to travel this distance, so messages should be processed within less time. Tests revealed that processing a message takes more than 700 µs, which would only allow for a resolution of around 210 km. In order to work with a TDoA scheme on the hardware present, a small but important change was introduced to the procedure.

Instead of sending all broadcast messages from the anchors at the same time t_{BR}, every anchor sends its broadcast at a previously determined time $t_{BR,i}$. Since all anchors are known to the master node, these times could be set at initialisation time and broadcasted to the client appropriately. Another way is to use a mathematical function to describe these times. A delay Δ_{BR} is introduced between the broadcasts of two anchors and the resulting broadcast time can be expressed iteratively

$$t_{BR,i+1} = t_{BR,i} + \Delta_{BR} \tag{2.13}$$
$$t_{BR,0} = t_{SI} + \Delta_{BR}, \tag{2.14}$$

or equivalently

$$t_{BR,i} = t_{SI} + (i+1) \cdot \Delta_{BR}. \tag{2.15}$$

Equation 2.15 is known to all anchors while equation 2.13 is used on the client. The delay parameter Δ_{BR} is provided as a network parameter with every synchronisation message from the master node and the anchor number i corresponds to its identifier (network address). This implies, that the TDoA values calculated from equation 2.12 need to be corrected by a multiple of Δ_{BR}. Using anchor 0 as reference, the values can be translated as if the broadcasts had happened simultaneously. The delay from anchor 0 to anchor i is $i \cdot \Delta_{BR}$ and the corresponding value thus

$$\tau_i = (t^{(c)}_{BR,i} - i \cdot \Delta_{BR}) - t^{(c)}_{BR,r}. \tag{2.16}$$

Sending a broadcast message underlies the same limitations as explained in section 2.2 regarding the precision of the sending timestamp (c.f. figure 2.2) and thus a complete broadcast from one anchor also includes two messages and the calculated times need to be adapted similarly.

From equation 2.16 it can be seen, that the whole broadcast process is running sequentially. A visual representation can be found using a time line (schedule), and is presented in figure 2.5. As soon as the last slot has ended, another synchronisation message can be sent and the whole process repeats itself. The interval between two synchronisation messages defines the synchronisation period T_{SI}. As all anchors need to have a time slot allocated, the synchronisation period has a lower limit

$$T_{SI} \geq (N+1) \cdot \Delta_{BR}, \tag{2.17}$$

where N is the number of anchors. The additional slot is used for the synchronisation message that initiates a new broadcast round.

2.3.3 Position

As explained in section 2.1.1, the location of the anchors is know at the time of initialisation of the network and each anchor is informed about its position before the network starts operating. The scheme assumes the position to be static, i.e. the anchors location will not change over time. Ideally, the position of all anchors would be broadcasted once and then used for every iteration of the localisation, but this would require every tag to be present and operational at the time this information is provided. To enable a client to join the network at any time, each anchor provides its own location with every broadcast it makes. The position is then stored together with its identifier and TDoA value on the

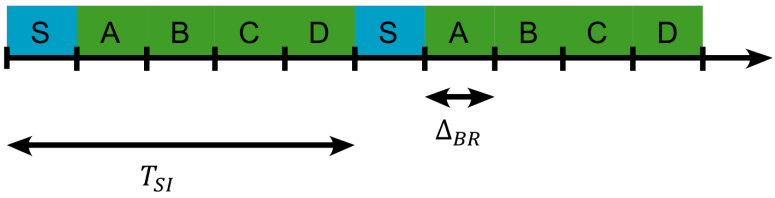


Figure 2.5: Visual representation of the broadcasting schedule using four anchors A-D. After the initial synchronisation message, every anchor uses its allocated timeslot of duration Δ_{BR} to send its corresponding broadcast message. The synchronisation also happens during an intervall of Δ_{BR} and the process is repeated after the slot of the last anchor has ended. The interval between two synchronisation messages defines the synchronisation period T_{SI}.

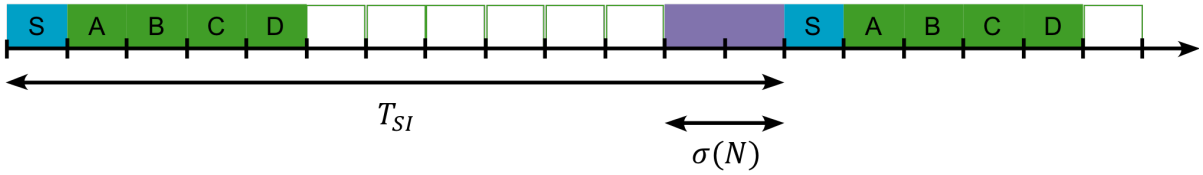


Figure 2.6: Visual representation of a complete network schedule using four anchors A-D and slots allocated for up to ten anchors. After the tenth slot, a slot dedicated for the calculation of the position is inserted. After the interval T_{SI}, the next broadcast round is initiated.

client until enough broadcasts are received to start the calculation of the clients position.

As the localisation algorithm takes a fair amount of time, the client needs to be left some time to do the necessary calculations. As the algorithm is computationally expensive, interrupting it is associated with large overhead and thus an interruption is not favourable. Due to the fact that it is running as a separate process on the client node, interruption must be prevented by suspending all communication inside the network. This is achieved by using a timeslot in the schedule dedicated to these calculations. Once all anchors have sent their respective broadcast, a slot is inserted before the master node sends the next synchronisation packet. The time needed for the calculation depends on the complexity of the optimisation problem, which increases with the number of anchors considered. The time is denoted by $\sigma(N)$ and a slot of that length is added at the end of the broadcasting slots. Figure 2.6 shows a general schedule, using four anchors A-D and slots allocated for up to ten anchors, together with the synchronisation and the positioning slot.

If the $\sigma(N)$ also needs to be considered, then the synchronisation period is also influenced by it, leading to a new lower bound

$$T_{SI} \geq (N+1) \cdot \Delta_{BR} + \sigma(N). \tag{2.18}$$

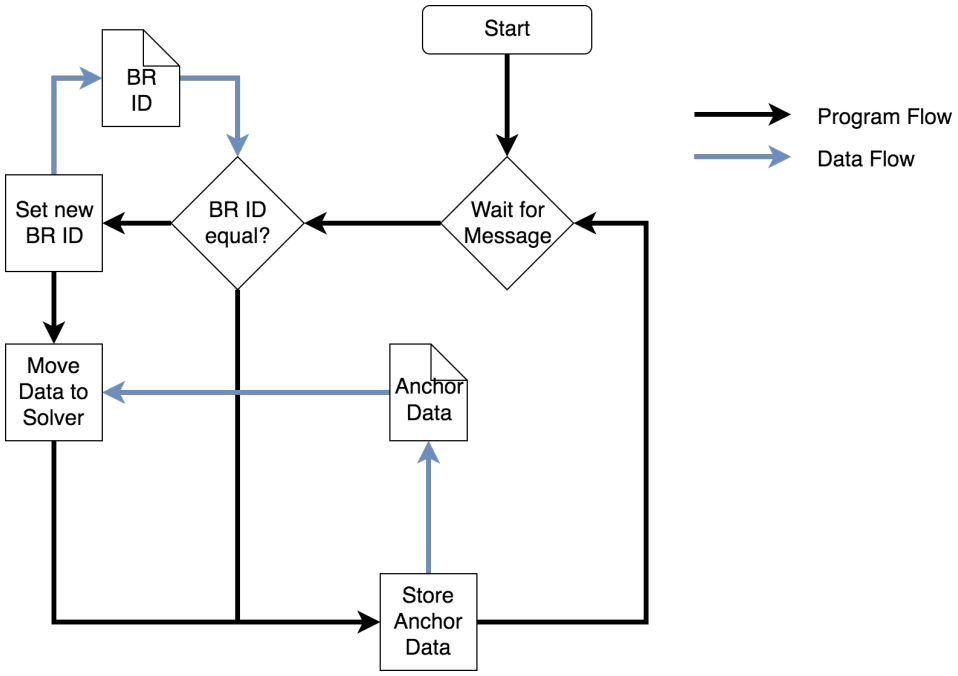


Figure 2.7: Visualisation of the data flow from the reception of a broadcast message to its use in the localisation process. The **Broadcast ID** (BR ID) is stored to assign the received broadcast to a specific round. Once a new round begins, the accumulated data set is provided to the solver and replaced with the new incoming data. If the id corresponds to the current round, then the data is just appended to the existing data set.

2.4 Positioning

As mentioned in section 2.3.1, the broadcasts of at least four anchors need to be received before the localisation algorithm can be initiated. Every broadcast round has an associated id, the *group id*, that is incorporated into every anchors message. After a new id has been received, the old messages are provided to the localisation algorithm an a new round of broadcasts can be stored. If enough messages have been captured, the localisation algorithm will start within its dedicated time slot of the broadcasting schedule. Further details of the localisation algorithm can be found in section 3, and a schematic of the data flow is depicted in figure 2.7.

Chapter 3

Localisation

If both the anchor i and reference anchor r send a message at the exact same time, because of their different distances to the client, these packets will reach the client at different timestamps. The delay/latency in the arrival of one packet with respect to the other is then defined as the TDoA value τ_i of anchor i.

By using the speed of light c in vacuum, one can calculate the corresponding distance difference of arrival ω_i (see figure 3.1):

$$\omega_i = \|\mathbf{A_i} - \mathbf{X_c}\| - \|\mathbf{A_r} - \mathbf{X_c}\| \tag{3.1}$$

Where $\|.\|$ is the 2-norm ($\|\mathbf{A} - \mathbf{B}\| = \sqrt{(x_A - x_B)^2 - (y_A - y_B)^2 - (z_A - z_B)^2}$).

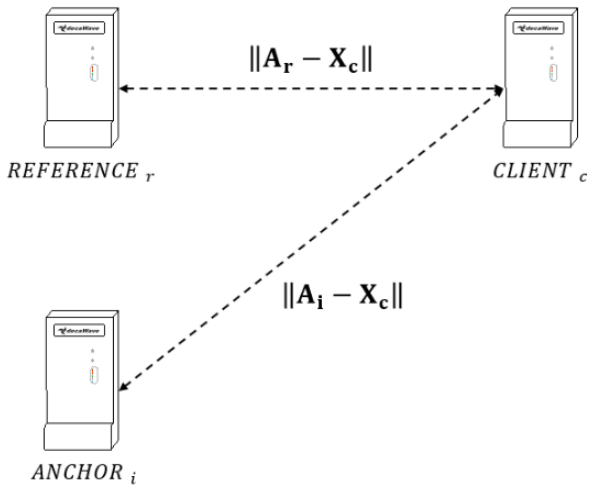
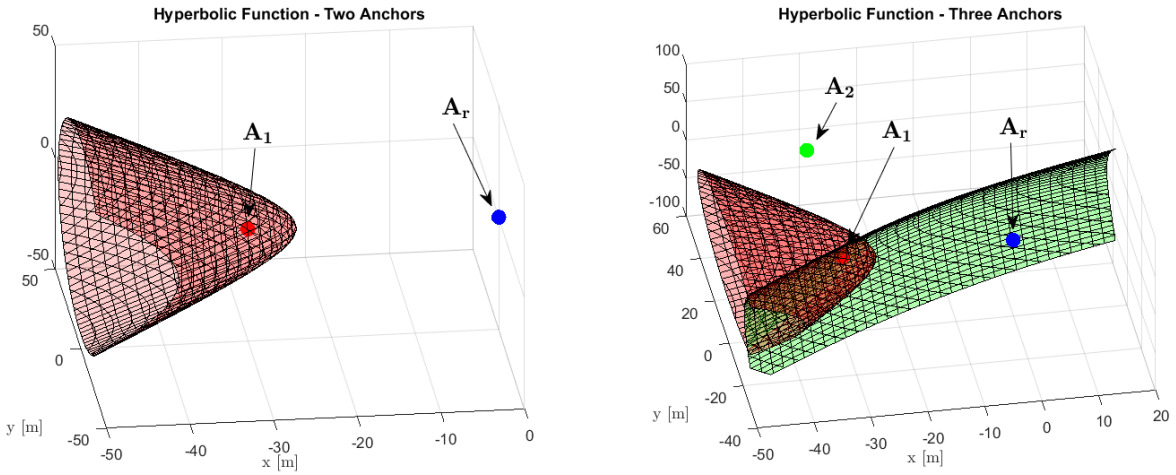


Figure 3.1: Schematic showing the distances of a reference anchor r and anchor i with respect to the client. Their difference represents the Distance Difference of Arrival ω_i

It is more convenient to work with distance differences, as we drop the 10^{-9} factor from the TDoA values (since they are given in the nanoseconds range). This can be a major advantage when considering the error generated by multiple arithmetic operations related to the computer's precision (a 32 bit micro-controller in this case).

If we look at the expression for a constant distance difference of arrival ω_i, one can see that the expression represents a one sided hyperboloid for the client's position as shown



(a) Hyperbolic function - two anchors

(b) Hyperbolic function - three anchors

Figure 3.2: Hyperbolic function of $\mathbf{A_1}$ and $\mathbf{A_2}$ using $\mathbf{A_r}$ as reference. In (b), the possible client positions would be given by the intersection of both hyperboloids.

in figure 3.2. That is, if the position of all devices and their respective distance difference of arrival are given, all points on the hyperboloid shown are going to satisfy the same expression (3.1) and can be a valid position for the client.

This shows that it is not possible to uniquely determine the client's position by using only two anchors. Uniqueness is only achieved by intersecting at least three hyperboloids, totalling four anchors: one as a reference and the other delivering three hyperboloids, whose intersection will give the unique position.

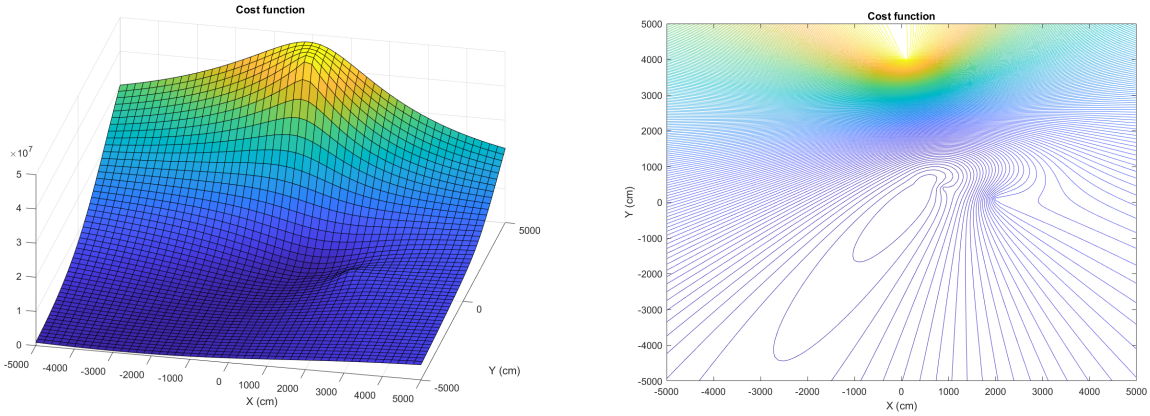
3.1 Minimisation Problem

Considering the measured distance difference values $\omega_i^{(m)}$, the pool of possible positions for the client can be further minimised by intersecting all hyperboloids found, as shown in figure 3.2 (b). For more than three anchors, this would yield a single point in space, assuming perfect knowledge of the TDoA data. But because of noise and errors in the calculated TDoA values, in general no point will be intersected by all hyperboloids of an overdetermined system (considering a non-ideal world). Therefore, a minimisation of the total error is calculated.

The error with respect to a calculated position $\mathbf{X_c}$ can be defined as

$$e_i = \omega_i^{(m)} - \omega_i = \omega_i^{(m)} - \|\mathbf{A_i} - \mathbf{X_c}\| + \|\mathbf{A_r} - \mathbf{X_c}\|\,, \tag{3.2}$$

where ω_i is the true distance difference for the given clients position $\mathbf{X_c}$.
The objective is to find the client's position $\mathbf{X_c}$ that minimises this error for all anchors



(a) Cost function for $z = 0$

(b) Contour plot of the cost function

Figure 3.3: Cost function for a four anchor system ($z = 0$) where anchors are placed on a random position on the xy-plane within the given range. The client's position $\mathbf{X_c}$ can be determined by finding the local minimum of the function.

A_i. The non-linear least-squares to be solved is therefore

$$\min_{\mathbf{X_c}} \sum_{i}^{N} e_i^2 = \min_{\mathbf{X_c}} \sum_{i}^{N} (\omega_i^{(m)} - \|\mathbf{A_i} - \mathbf{X_c}\| + \|\mathbf{A_r} - \mathbf{X_c}\|)^2 \tag{3.3}$$

Figure 3.3 shows a possible plot for the cost function at $z = 0$. In this case, four anchors were placed at random positions and noise added to the real TDoA values. No noise was added to the position of the anchors, i.e., the positions are considered to be perfectly known.

By studying both plots it is possible to localise the minimum of the function, where theoretically, the client was placed. Looking at it by eye this would be close to the origin $(0, 0)$. In this particular simulation the client was positioned at $(2\,\text{cm}, 0\,\text{cm})$ with $z = 0$ (making it a 2-D problem).

3.1.1 Solving the Optimisation Problem

Of the different iterative and non-iterative algorithms available for solving such problems in embedded devices, two came into question in this project: an iterative Levenberg-Marquardt algorithm and a non-iterative algebraic solver (see section 3.2).

In the case of the iterative Levenberg-Marquardt algorithm, which is an interpolation between the conventional Gauss-Newton and a steep descent algorithm, the iteration step is defined as follows:

$$\mathbf{X_c}^{(s+1)} = \mathbf{X_c}^{(s)} + (\mathbf{J_e}^T \mathbf{J_e} + \lambda \cdot \mathbf{I})^{-1} \mathbf{J_e}^T \mathbf{e}(\mathbf{X_c}^{(s)}) \tag{3.4}$$

where λ is the damping factor[1] and $\mathbf{J_e}$ is the Jacobi-matrix of the error e with respect to the variables of $\mathbf{X_c}$ (which are the Cartesian coordinates x, y, z):

$$(\mathbf{J_e})_{ij} = \frac{\partial e_i(\mathbf{X_c}^{(s)})}{\partial X_{c[j]}}, \text{with } j = x, y, z \tag{3.5}$$

[1] Note that $\lambda = 0$ leads to the conventional Gauss-Newton algorithm.

Applied to our residual function e_i:

$$(\mathbf{J_e})_{ij} = \frac{A_{i[j]} - X_{c[j]}}{\|\mathbf{A_i} - \mathbf{X_c}\|} - \frac{A_{r[j]} - X_{c[j]}}{\|\mathbf{A_r} - \mathbf{X_c}\|} \tag{3.6}$$

$$\mathbf{J_e} = \begin{bmatrix} \frac{x_{A_1}-x_c}{\|\mathbf{A_1}-\mathbf{X_c}\|} - \frac{x_{A_r}-x_c}{\|\mathbf{A_r}-\mathbf{X_c}\|} & \frac{y_{A_1}-y_c}{\|\mathbf{A_1}-\mathbf{X_c}\|} - \frac{y_{A_r}-y_c}{\|\mathbf{A_r}-\mathbf{X_c}\|} & \frac{z_{A_1}-z_c}{\|\mathbf{A_1}-\mathbf{X_c}\|} - \frac{z_{A_r}-z_c}{\|\mathbf{A_r}-\mathbf{X_c}\|} \\ \frac{x_{A_2}-x_c}{\|\mathbf{A_2}-\mathbf{X_c}\|} - \frac{x_{A_r}-x_c}{\|\mathbf{A_r}-\mathbf{X_c}\|} & \frac{y_{A_2}-y_c}{\|\mathbf{A_2}-\mathbf{X_c}\|} - \frac{y_{A_r}-y_c}{\|\mathbf{A_r}-\mathbf{X_c}\|} & \frac{z_{A_2}-z_c}{\|\mathbf{A_2}-\mathbf{X_c}\|} - \frac{z_{A_r}-z_c}{\|\mathbf{A_r}-\mathbf{X_c}\|} \\ \vdots & \vdots & \vdots \\ \frac{x_{A_n}-x_c}{\|\mathbf{A_n}-\mathbf{X_c}\|} - \frac{x_{A_r}-x_c}{\|\mathbf{A_r}-\mathbf{X_c}\|} & \frac{y_{A_n}-y_c}{\|\mathbf{A_n}-\mathbf{X_c}\|} - \frac{y_{A_r}-y_c}{\|\mathbf{A_r}-\mathbf{X_c}\|} & \frac{z_{A_n}-z_c}{\|\mathbf{A_n}-\mathbf{X_c}\|} - \frac{z_{A_r}-z_c}{\|\mathbf{A_r}-\mathbf{X_c}\|} \end{bmatrix}$$

The implementation of the algorithm is quite simple. By having the *data* structure with all the anchor positions $\mathbf{A_i}$ and measured distance difference values w_i in the following format,

$$data = \begin{bmatrix} x_{A_1} & x_{A_2} & \dots & x_{A_{n-1}} & x_{A_r} \\ y_{A_1} & y_{A_2} & \dots & y_{A_{n-1}} & y_{A_r} \\ z_{A_1} & z_{A_2} & \dots & z_{A_{n-1}} & z_{A_r} \\ w_1 & w_2 & \dots & w_{n-1} & 0 \end{bmatrix}$$

a possible implementation is given by **Algorithm 1**.

Algorithm 1 Levenberg-Marquardt

```
function SOLVER(data, init, iter, lambda)
    ## Initial values (init is a 3x1 vector)
    x_k ← init
    for count := 1 to iter step 1 do
        for i := 1 to na step 1 do
            ## Get the distances between anchors and current position
            dist(i) ← norm(data(1:3, i) − x_k)
        end for
        for i := 1 to na − 1 step 1 do
            for j := 1 to 3 step 1 do
                ## Populate Jacobian
                J(i, j) = (data(j, i) − x_k(j))/dist(i) − (data(j, na) − x_k(j))/dist(na)
            end for

            ## Populate residue (error)
            err(i) = data(4, i) − dist(i) + dist(na)
        end for
        x_k ← x_k − (J^T J + lambda · I)^{-1} J^T err
    end for
    return x_k
end function
```

3.1.2 Initial Condition and Damping Factor

The initial condition *init* used by the solver plays an important role on the correct convergence to the local minimum of the cost function and the number of iterations needed to

do so. To enhance the algorithm's effectiveness with the hope of minimising the number of iterations needed, a non-zero initial condition is fed to the solver. In this case the mid-range of all anchors (the arithmetic mean of the maximum and minimum of their positions):

$$init_{[j]} = \frac{1}{2}(\max_i A_{i[j]} + \min_i A_{i[j]}), \text{where } j = x, y, z \tag{3.7}$$

This approach is only efficient if no prior position data is available yet, as it is assumed that the client is somewhere in-between the anchors. As soon as the solver returned at least one valid position, the last recorded data point can be used as the initial condition. With high refresh rates (solving of the problem), the client is expected to be close to the last calculated position and a convergence is expected to happen much faster.

As for the damping factor, any value different from zero (which leads to the conventional Gauss-Newton algorithm) showed to be effective. Many simulations were performed using different setups (number of anchors and their position, amount of noise added to the TDoA values, etc.) for different damping factors $\lambda \in [0, 1]$. To find the best fit for our application, no mathematical and physical characteristics were studied to find λ's impact on the effectiveness of the solver. It was purely tuned by simulations. As for now, $\lambda = 0.1$ is used.

3.1.3 Evaluating the Solver

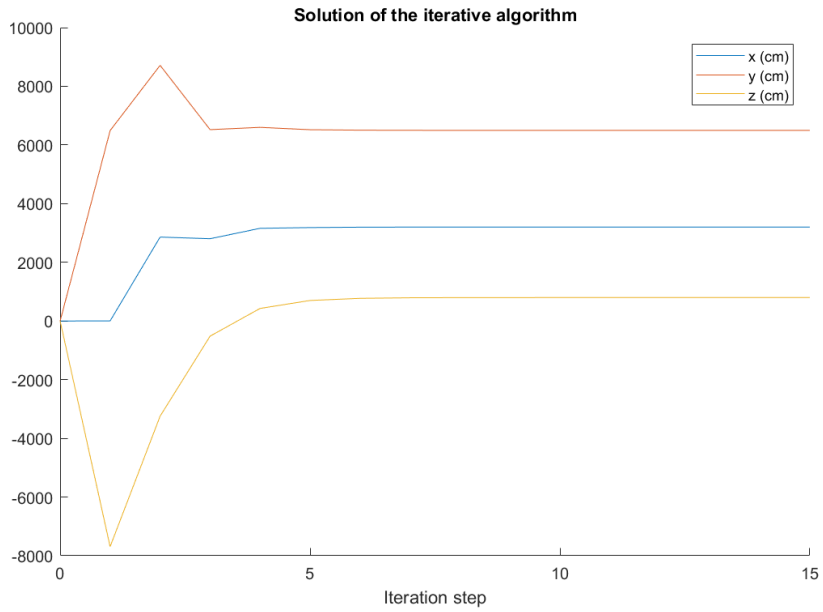
Figure 3.4 shows a possible evolution of the output position given by the solver for each iteration step. As can be seen in (a) and (b), each iteration brings the resulting position closer to the minimum of the cost function (contour plot (b)).

According to the many simulations performed using different setups (noise added, anchor/client positions, etc.), an average of five to eight iteration steps are needed to get a reasonable position. These simulations were performed using only the mid-range concept introduced in section 3.1.2. Using the last recorded position as the initial condition for the case of a dynamically moving client, this number may be lowered even further.

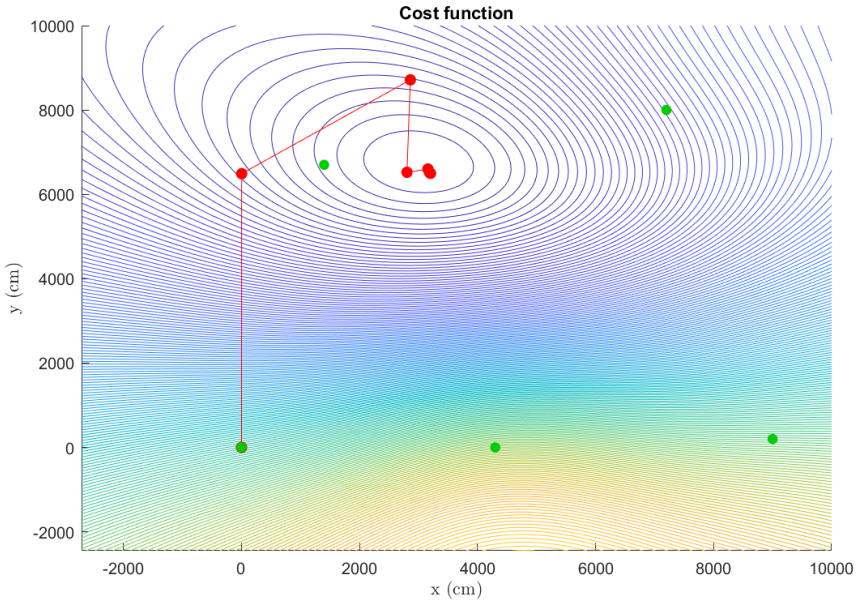
Since the micro-controller used may take up to 1.5 ms per iteration step to solve the problem (according to simulations using four anchor points), a default constant number of 10 iterations is set for the current system (this parameter may be changed in the system configuration - see section 4.2.1). Note that this implementation does not have a break condition. The number is constant, independent of the quality of a given position for a given iteration step.

3.2 Algebraic Solver

Besides the iterative algorithm mentioned in section 3.1.1, a non-iterative algebraic solver was tested. The advantages of such a solver is the computation time needed. The actual



(a) Simulation - Plot of the solver in action



(b) Simulation - Contour plot of the cost function

Figure 3.4: One of the many simulations performed using the iterative algorithm. In this case the initial condition is given by the origin $(0, 0, 0)$. (a) Plot of the evolution for each component (x, y, z). (b) Evolution of the position (in red) plotted on a contour plot of the initial cost function ($z = 0$). The green dots represent the positions of the anchors.

implementation of the iterative algorithm needs up to 10 ms of dedicated processing time in order to find a reasonable position. An algebraic solver would drastically decrease this computation time, making it possible to increase the overall refresh frequency at which position data is delivered.

The algorithm in question was developed by Stephen Bancroft with an attempt of solving the GPS equations with an algebraic solver [3]. It showed to be very effective with perfect data, giving positions with the same accuracy as the Levenberg-Marquardt algorithm in most cases. However it failed to do so with noisy TDoA data. With noise up to 10 cm (which may be expected according to the DW1000's datasheet), the calculated position was completely unpredictable and random, giving output position values that were up to kilometers apart from the actual position.

With data that does not contain much noise, this would certainly be a good approach, as the computation time is indeed very low compared to other algorithms (according to the simulations done). But because the hardware currently used cannot deliver the accuracy needed to make the algebraic solver reliable enough, it is not implemented in the current implementation.

3.3 Two-Way-Ranging (TWR)

Two-Way-Ranging is a two-way communication ranging procedure that takes place between two independent devices. The idea is straightforward: by sending and receiving packets, one can calculate the time it took for the packet to go from one node to the other and back. Thus, the distance can be determined by knowing the propagation speed. Since this assumes that both devices are actively sending packets back and forth, this will not be the focus of the positioning system studied here.

TWR comes in hand when initialising the whole system. The localisation using TDoA assumes all positions of the anchors to be known. Instead of hard-coding all these positions, the Master requests a TWR between all anchor pairs in order to get all relative distances. Using this information, the position of each individual anchor can be estimated.

Chapter 4

Hardware and Software

This chapter discusses the hardware and software used to implement and debug the system mentioned above. It also describes the methods used to counteract the problems regarding offset shift/drift, for example, and how the available features of the hardware were applied in order to get all the data needed for a precise and fast localisation.

4.1 Device Nodes

One interesting aspect of the system is that the device nodes (master, client and anchor points) all share a common piece of hardware. They are all based on the same DecaWave Development Board DWM1001-DEV and differ only in the firmware. Figure 4.1 shows the front view of the DWM1001-DEV module used, which is equipped with an UWB module (DW1000) and the nRF52 chip as the host micro-controller.

The main features of this hardware combination are listed below:

- Bluetooth capability
- J-Link probe for debugging purposes
- 8 Mbps SPI communication with the UWB DW1000 module
- Crystal-trimming for fine clock speed adjustments
- A double-buffer to cope with the high rate of incoming UWB packets

The board is also equipped with many other features like multiple status LEDs, user buttons, serial ports for communication etc.

4.1.1 Clock Offset & Drift

As mentioned in section 2.2.1 and 2.2.2, one has to account for the differences in clock rate and offset of the two communicating devices in order to properly calculate a TDoA value. This showed to be very hard, as the clocks come with an uncertainty in their counting (due to the clock accuracy). They have a ppm (parts-per-million) property which gives

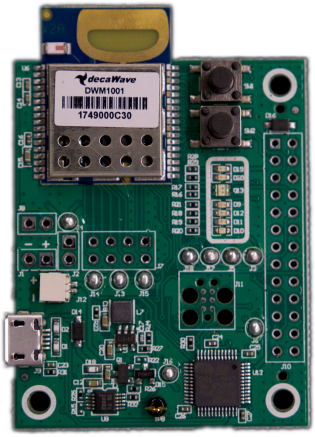


Figure 4.1: The complete DWM1001-DEV module with an UWB chip and a nRF52 chip as the host MCU

the time in cycles a clock deviates from its actual frequency after one million clock cycles. This gives rise to the complexity of calculating the real offset and drift as no two clocks will be ticking at the same frequency.

Three major approaches were used to counteract the effects of different clock speeds:

Crystal trimming The nRF52 comes with a feature which allows so-called crystal trimming. It gives the board the capability of adjusting the capacitances parallel to the oscillating crystal (which meditates the clock speed). By changing these capacitances in discrete steps, one can make the clock run faster or slower. As the Master node has the task of giving the reference time, all anchors will dynamically change their crystal-trim properties after each sync packet, making the clock go faster or slower until their relative drift with respect to the Master is minimised.

Modelling of the drift The available discrete trim values are very limited, so it is not possible to completely get rid of the drift with this feature. Their contribution to the recorded timestamps are further handled by software. The drift is modeled as being a constant. The offset will consequently have a linear temporal dependency, whose slope is given by this drift:

$$\Delta_i(t) = \kappa_i \cdot t + \Delta_i(0) \tag{4.1}$$

The reason crystal trims were used to minimise the drift, although it is simply modelled as a constant, is that it makes the deviations for this time correction much more subtle for the case of small slopes.

Sync period Since the contribution of the linear modelling mentioned above is only updated after a sync packet, increasing the frequency at which these updates happen may increase the accuracy of the correction term. 50 ms showed to be a very good value for the synchronisation period, but it may be tuned according to other characteristics of the system (like broadcast interval, for example - see section 4.2.1)

4.1.2 Double-buffer

Because of the high frequencies at which broadcast packets have to be sent, the client may not be able to serially read all packets in time without missing data. The speed at which the UWB module communicates with the host controller is very low (8 MHz, way below the supported 20 MHz of the DW1000 chip). Many packets may arrive at the DW1000 before prior stored data is completely transferred to the host controller. Old data will then be partially overwritten by the time it is transferred to the host controller, which then receives corrupted data.

The solution to this problem is to activate the double-buffer of the UWB chip. Its purpose is to write the information in to one of the buffers, while the host controller reads from the other. As soon as the data has successfully been transferred, the buffer is freed and can now receive new arriving packets, while the host now reads from the first buffer.

4.2 System Interface

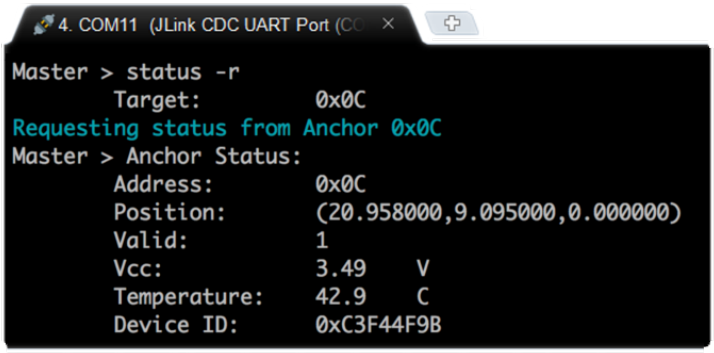
The overall system is very complex, with many different parameters mediating refresh rates, sync periods, anchor data and positions and much more. The sections below discuss the many interfaces implemented to make the experience of configuring and interpreting all the data more user-friendly.

4.2.1 System Configuration

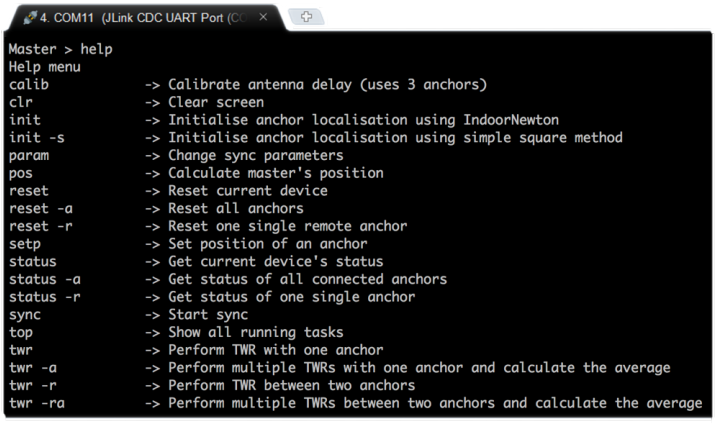
One major concern with respect to the system's complexity is the amount of tunable parameters that control the whole system. If one of these parameters is not set correctly, other features may also be affected. A more convenient way is needed in order to change these values in a fast and reliable manner, as hard-coding these values simply takes too long and goes against the concept of ease-of-use (uploading the code and setting up the system every time a hard-coded value is changed takes too much time).

The solution was the creation of a simple but helpful command-line interface that enables the user to easily change any parameter of the system during execution. Figure 4.2 (a) shows an example for the *remote status report* command which gives all the details for a given anchor point. This includes hardware details (voltage, temperature), anchor position, device identifier and much more.

This command-line also gives the possibility of taking full control over the anchors, like performing a hardware reset. This can be useful in the case that the Master node loses all anchor information (because of an external reset, for example). It resets the anchors in order for them to resend their previous IDs.



(a) Command for status report



(b) Interface's help menu

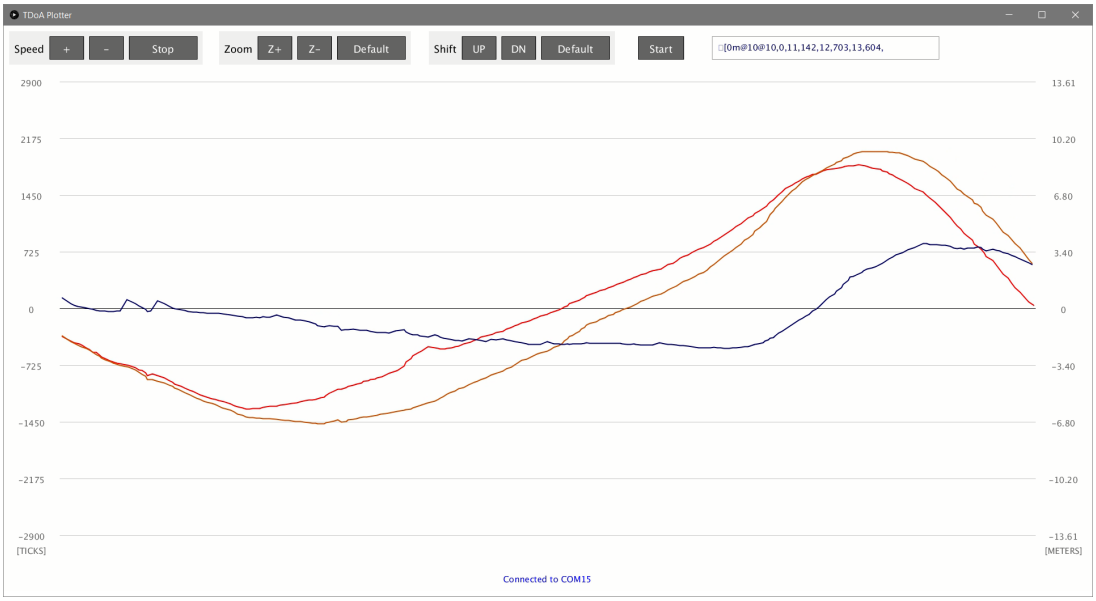
Figure 4.2: Command-line interface for configuring the system. (a) remote status report that reads detailed anchor information (here for anchor 0x0C). (b) Help menu with all commands supported (master node).

4.2.2 Graphical User Interfaces

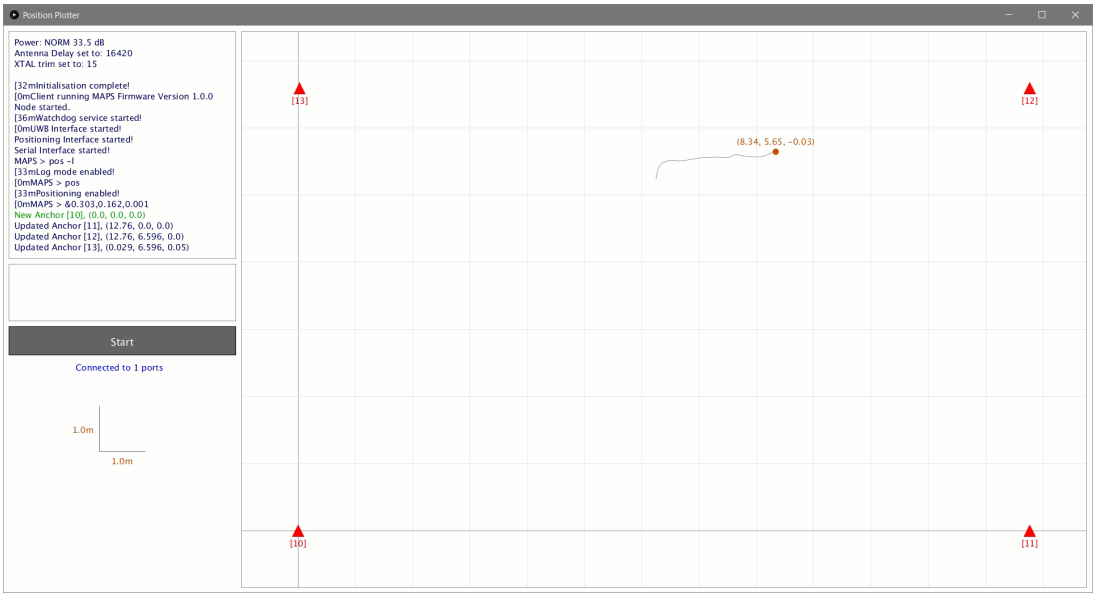
To debug the received distance differences of arrival and calculated position of the client, two user interfaces were created (See figure 4.3). One Graphical User Interface (GUI) plots only the TDoA values of the anchors and the other plots the resulting estimated position of the client. The broadcast of the TDoA/position data is activated by sending the *pos [-l]* or *tdoa* command to the Client connected to the PC. This triggers a thread that sends debugging information through the serial port, in human or machine readable text formats.

The GUIs also support multiple devices connected to the same computer. The information of all devices are all mapped and plotted on the same map canvas but with different colors and upper ID text (in order to distinguish them). Besides that, the following features come with the software:

- Anchor position with printed individual ID
- Client position with printed individual ID
- Line collection with client path history (last 2 seconds)
- Grid with scale for dimensioning
- Logger with debug information
- Start/Stop button to activate/deactivate plotting



(a) User Interface for TDoA values



(b) User Interface for position

Figure 4.3: User Interfaces used to debug and graphically represent the data received. (a) Plots the TDoA values for all anchors, giving the time/distance differences in DecaWave ticks and meters (b) Plots the estimated position of the client together with the position of all the anchors.

Chapter 5

Conclusion

The findings of this project confirmed UWB as a reliable and accurate source of localisation. Furthermore, the benefits of a TDoA scheme could be utilised, in order to achieve the desired goals. Additional considerations can be made with respect to accuracy, speed, robustness and the ease of use. A few options will be considered below.

5.1 Clock Synchronisation

As mentioned in section 2.2, the accurate synchronisation of the clocks used on individual devices affects the localisation accuracy crucially. In this setup, a global time frame is established by using the clock of the unique master node as a reference in terms of offset and clock rate. This procedure proved to be accurate enough to meet the requirements for autonomous video production, but cannot deliver the accuracy a vision based system offers (cf. [2]). In order to further improve localisation accuracy, a more robust clock synchronisation could be established, which can be achieved in multiple ways.

5.1.1 Oscillator

One approach would be to improve the accuracy and/or resolution of the on-board oscillator used on the DW1000 Module. The datasheet [4] specifies an oscillator with a resolution of 64 GHz with several ppm (parts-per-million) offset. Assuming a worst case scenario of ± 3 ppm, this would generate a relative offset of 3 µs after only 1 s, corresponding to a distance error of $\sim$900 m! Without periodic clock synchronisation, including drift, a reliable localisation would not be possible. Additionally, with a nominal frequency of 64 GHz, the distance resolution is limited to $\sim$4.7 mm. Using an oscillator with higher resolution would increase spatial resolution, whereas higher accuracy (lower ppm offset) allows for fewer synchronisation messages and thus higher broadcast rate. For millimeter resolution, a clock rate of $\sim$300 GHz would be necessary, the clock rate can be stabilised by using oscillators less sensitive to external influences, such as temperature or voltage.

5.1.2 External Clock

Using an external device dedicated to timekeeping allows more sophisticated methods to be applied to clock synchronisation. Using atomic clocks for example would allow them to

be synced once and the network would maintain accuracy over extended periods of time. This approach however is much more costly and the clock itself would exceed the size and energy consumption of the device by orders of magnitude. An alternative approach involving atomic clocks would make use of the time synchronisation capabilities of the GPS by attaching a receiver to every node. On the other hand, this would require all nodes to be able to receive such signals, which is an assumption disagreeing with the desired application scenarios of the system.

5.1.3 Clock Synchronisation Algorithm

The third, and most effective, option would be to use a sophisticated clock synchronisation algorithm. Algorithms such as *Christian's Algorithm* or *The Berkeley Algorithm* [5], both being successfully used in IP-Networks could offer a more reliable timing performance. Since the whole network communication is based on similar principles as an IP-Network, adaptions of such algorithms can be used to try to improve accuracy.

Another approach is to use TDoA measurements, which would be an interesting approach, since the network architecture is designed for such measurements. Tests performed on similar hardware by McElroy et al. suggest different protocols in order to synchronise clocks in a distributed sensor network [6].

5.2 Hardware

While testing the system for performance, two limiting factors where discovered. The biggest limiter was found to be the solving of the optimisation problem $\sigma(N)$ (also cf. section 2.3.3 and equation 2.18). With a minimum number of anchors (4), the algorithm already takes $\sim$15 ms to complete, whereas the corresponding sync and broadcast messages only take up $\sim$4.5 ms combined! Using on-board calculations, the positioning frequency is thus limited to around 50 Hz, although the theoretical limit without solving the optimisation problem would allow positioning with up to 220 Hz. The use of dedicated hardware to solve the optimisation problem, such as FPGAs[1] or ASICs[2] is thus a reasonable consideration. Apart from solving the problem faster, they are more energy efficient compared to solving the problem in software.

The second limiter is the speed at which messages can be sent. Since every localisation-relevant message (sync and broadcast) consists of two packets, the time it takes to prepare and send these also influences the positioning rate. Considering equation 2.18, the factor Δ_{BR} is limited by this time. If the sending of the two messages is denoted by ϑ_m, the limit can be found to be

$$\Delta_{BR} \geq \vartheta_m + \max_{i,j} \hat{t}_{i,j}, \tag{5.1}$$

where $\max_{i,j} \hat{t}_{i,j}$ corresponds to the longest time of flight between two nodes. The time of flight can only be reduced by decreasing the longest distance between any two nodes,

[1] field programmable gate array
[2] application-specific integrated circuit

which is not always a viable option. The other possibility to decrease Δ_{BR} is to decrease ϑ_m, which was found to be around 700 µs. Considering the data rate offered by the devices used [4, 7], this time could be lowered, as the DW1000 IC exceeds the I2C[3] speed of the host IC. If the speed of the host IC would match the one of the DW1000, smaller delays between preparing and sending a message could be achieved. If a transceiver was used, that allows to incorporate the timestamp of sending into a packet itself, the second packet would be superficial. This is expected to half the time of sending a message ϑ_m.

5.3 Network Layout

The current architecture of the network builds on the assumption, that the master node is always in reach of any node connected. Furthermore to reliably synchronise clocks, all devices (except client nodes) have to be stationary. Using considerations from section 5.1.3, a refined clock synchronisation algorithm would allow the master node (in this scenario acting as a time server) to be out of direct sight (prohibiting a direct communication link) to some of the other nodes. This would require the network to be able to relay messages through it, similar to a computer network, and every node having at least one path to the master. Removing the restriction of direct connection would also allow the devices to communicate over a different channel for administrative purposes, such as Ethernet. Network administration messages and status reports could be transmitted over the secondary channel to avoid interruption of the broadcasting schedule, increasing the throughput of localisation-relevant messages. Another advantage is a more complex geometry of the space covered by the network. Instead of having one large open space, multiple rooms or buildings can be reached within the same network.

Alternatively, the network could be configured in a way, that allows multiple networks to be present and distinguishable. This would enable them to operate in close proximity of one another without interfering and covering multiple applications, having different requirements on localisation speed.

[3] inter-integrated circuit; synchronous bus used to communicate between devices.

Bibliography

[1] GPS NAVSTAR - Department of Defence, United States, *Global Positioning System Standard Positioning Service Performance Standard*, 4th ed. [Online]. Available: https://www.gps.gov/technical/ps/2008-SPS-performance-standard.pdf.

[2] Y. Bosshard, M. Rogenmoser, and T. Salzmann, "Controlling quadcopters using ultra-wideband," ETH Zurich, 2018.

[3] S. Bancroft, "An algebraic solution of the gps equations," *IEEE Transactions on Aerospace and Electronic Systems*, vol. AES-21, no. 1, pp. 56–59, Jan 1985.

[4] "DWM1000 datasheet," https://www.decawave.com/sites/default/files/dwm1000-datasheet-v1.6.pdf, 2016, accessed: 2018-10-13.

[5] A. S. Tanenbaum and M. van Steen, *Distributed Systems Principles and Paradigms*. Prentice Hall, 2002.

[6] C. McElroy, D. Neirynck, and M. McLaughlin, "Comparison of wireless clock synchronization algorithms for indoor location systems," Decawave Limited, date unknown.

[7] "Dw1000 user manual," https://www.decawave.com/sites/default/files/dw1000_user_manual_2.12.pdf, pp. 225–232, 2017, accessed: 2018-10-13.