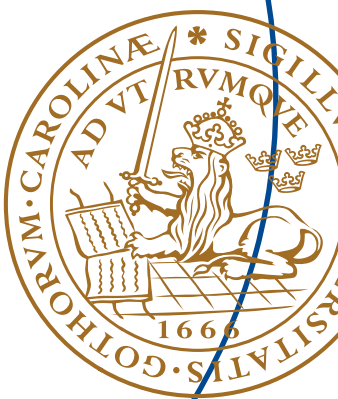
Master's Thesis

Multiobject localization using wavelet radar sensors

Per Atlevi



Department of Electrical and Information Technology,
Faculty of Engineering, LTH, Lund University, June 2014.
In cooperation with Acconeer AB.

Master thesis:
Multiobject localization using wavelet radar sensors

Author: Per Atlevi

Date: 6/23/2014

Abstract

In this Master Thesis, a good two-step positioning scheme based on Time of Arrival (TOA) measurements is found in order to localize objects, such as fingers, using sensors acting as both transmitters and receivers. A two-step positioning approach takes the input signals from the sensors and first estimates position related parameters, in this case TOA, and then uses these parameters to estimate the position. The scheme is implemented in an Android application. The input signals are generated and then transferred to the application via USB. Three Time Delay Estimation algorithms as well as four Localization, or Positioning, algorithms are evaluated. The first Time Delay Estimation algorithm is based on a simple peak detection approach, the second uses cross covariance and the third is a least squares estimator. The Localization algorithms use both geometrical and statistical approaches; the first uses trilateration, the second uses a form of weighted trilateration, the third is based on the Maximum Likelihood (ML) approach and finds the position using a gradient descent algorithm and finally the fourth also uses the ML approach but finds the position using a brute force search algorithm. All the algorithms are assessed based on stability, accuracy and computational complexity. They are first tested separately and then in conjunction with each other. In addition to this, sensor placement and density are investigated in order to minimize the estimation errors. The algorithms chosen to be implemented in the application are able to localize multiple objects with millimeter precision at high rates.

Preface

This Master Thesis started in February 2014 and was finished in June the same year. The majority of the work done in the Thesis was conducted at the research and development company Acconeer Wavelet Technologies in conjunction with the Electrical and Information Technology (EIT) department at Lund University.

The Work was done as a final step in order to get a Master's degree in Nanotechnology at the Faculty of Engineering at Lund University.

I would like to thank Mats Ärlelid and Mikael Egard from whom I received valuable help throughout the whole time working on the Thesis. I would also like to thank Fredrik Tufvesson, my examiner, for valuable input when writing the final report. Lastly, I would like to thank Andreas Jacobsson for all his help on the Time Delay Estimation part as well as the overall statistics in the paper.

Lund, June 2014
Per Atlevi

Table of Contents

1 Introduction

1.1 Problem Definition and Aim of the Thesis

The aim of this project is to investigate methods for identifying multiple objects based on input from multiple wavelet radar sensors. An appropriate method should then be implemented on an Android Smartphone.

A sensor transmits radar wavelets and then receives reflections of the same wavelets (assuming one or more objects are in range). The reflected signals received by the sensors are correlated with a reference signal producing output signals as can be seen in figure 1. The signals have thermal noise and since the correlation is (practically) analog, the overall noise can be assumed to be of Gaussian nature with zero mean (which is important for certain methods in this thesis, as can be seen in section 5.1.2). The sensors together form a sequential Single-Input and Single-Output (SISO) system. In a system like this, each sensor transmits and receives signals in turn, making it sequential as opposed to parallel. The approach used in this project is a two-step positioning procedure; first, estimation of position related parameters is done and second, the position estimation is done. The estimations are based on TOA measurements and hence, trilateration methods for positioning are mainly considered. One reason for using a two-step approach instead of a one-step approach (where the position estimation is done directly using the input signals) is that methods using the one-step approach in general have a higher computational complexity than methods using the two-step approach. But the main reason is that the resulting Android application should easily be able to present distance measurements (i.e. the first step in the two-step procedure).

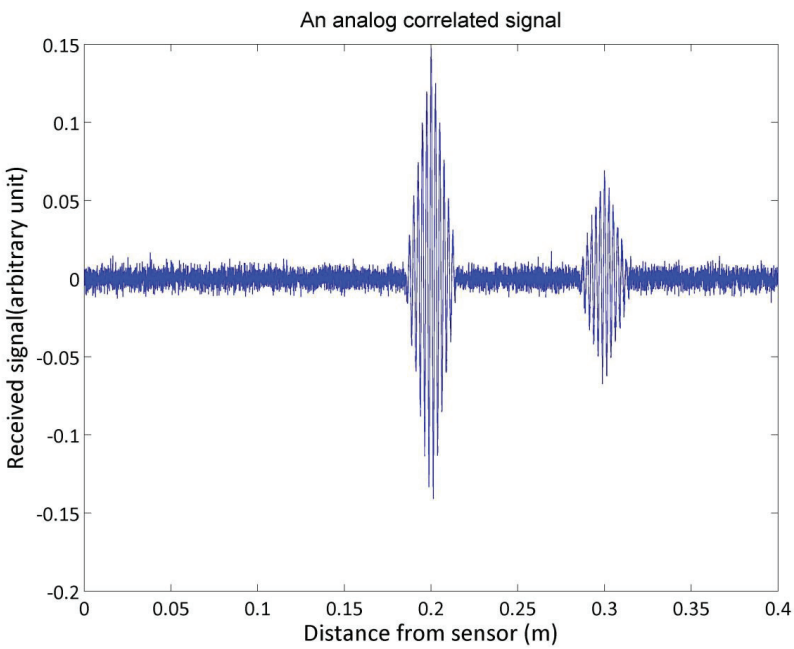


Figure 1. A correlated analog signal with reflections 0.2 m and 0.3 m away

All the input data used in this project will be generated in Matlab (i.e. the data used is simulated).

The sensors support ranges from 10 to 500 mm. When estimating an object's position in two dimensions, the sensors are distributed along a 1-dimensional, 0.1 m long array (more specifically along the x-axis) and the objects are located in an area of 0.157 m^2 where the restrictions on x and y are -0.1 m $\leq x \leq$ 0.1 m and 0.015 m $\leq y \leq$ 0.8 m. This area will be called the Area in the rest of this thesis. When estimating in three dimensions, the sensors are distributed within a plane (more specifically in the xy-plane) where the sides are 0.1 m and the objects are located in a volume of 0.0314 m^3 where the restrictions on x, y and z are -0.1 m $\leq x \leq$ 0.1 m, -0.1 m $\leq y \leq$ 0.1 m and 0.015 m $\leq z \leq$ 0.8 m. This volume will be called the Volume in the rest of this Thesis.

With the above prerequisites the following should be investigated:

- Three methods for time delay estimation is to be investigated more closely, namely:
  a) Peak Detection
  b) Covariance
  c) Least Squares
  The goal is to find the best one considering stability, accuracy and the fairly limited computational power in an Android application. The reason that these three should be investigated is mainly due to their quite low computational complexity.

- Considering the constraints on sensor placement, how should the sensors be placed to minimize the position estimation error? To be investigated is mainly:
  a) How far apart the sensors should be
  b) How many sensors that is ideal
  alternatively, is there a way to minimize the impact of suboptimal sensor placement?

- Which of the following methods is the best for estimating the position of 1 object in either 2 or 3 dimensions, considering a tradeoff between accuracy, stability and computational power?
  a) Linear Least Squares
  b) Non-Linear Least Squares
  c) Centroid
  d) Weighted Centroid
  also, which of these are applicable for estimating multiple objects?

- Given that you have a certain error in the time delay estimation, Δd_i, how big of an impact does this have on the error, $\Delta x, \Delta y, \Delta z$, in position estimation in 2 and 3 dimensions?

- If appropriate methods for time delay estimation as well as position estimation are found, how do you effectively implement these in Java (used in Android)?

1.2 Tools

The main tools for finding different papers and reviews on Time Delay Estimation and Localization/Positioning have been IEEE Xplore and Google Scholar.

The algorithms studied have been implemented and evaluated in Matlab R2013b.

The implementations of algorithms in Java as well as the creation of the application have been realized in the open source tool Eclipse IDE. And ADT (Android Development Tools) plugin has been used in Eclipse to easier create the Android interface in the application. The application has been tested on a Sony Xperia ZL.

1.3 Disposition

The rest of this paper is organized as follows. In the next section, section 2, the background is presented. In section 3, the time delay estimation is considered and in section 4, sensor distribution analysis is performed. In section 5, the different positioning algorithms for one object as well as some error analysis is considered. In section 6 and 7, multiobject localization and the whole two-step positioning estimation is discussed, respectively. Section 8 handles the Android application part of the Thesis and finally section 9 briefly states the conclusions.

2 Background

Localization has been a very important topic for several years. Localization enables a range of applications such as GPS, target tracking, environmental monitoring, robotic mapping, emergency interventions and node positioning in wireless sensor networks [1][2][3]. For a long time, localization was mainly realized in 2 dimensions, but in recent years the focus has been drifting more towards 3-dimensional localization. Localization can also be referred to as positioning.

There are several different ways to locate, for example, a target sensor in a sensor network. Firstly, there are two main approaches, namely direct positioning and two-step positioning. The direct positioning approach takes received signals from the sensors in the network and directly estimates the position of the object. The two-step positioning approach, however, first estimates a number of position related parameters based on the received signals and then estimates the position of the object based on these parameters, see figure 2.

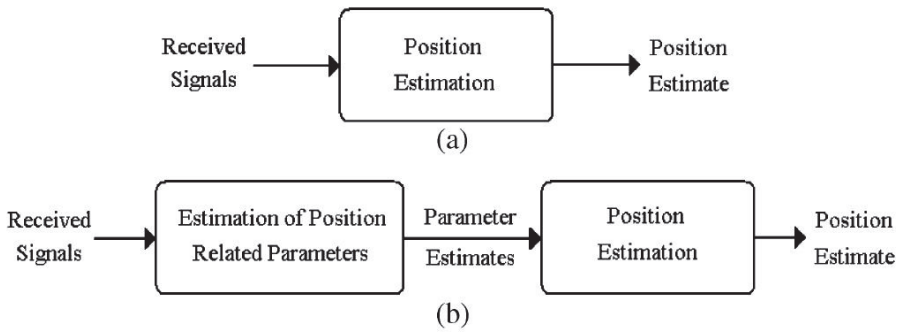


Figure 2. (a) Direct positioning approach and (b) Two-step positioning approach [4]

2.1 Direct Positioning

As mentioned above, direct position determination (DPD) is a one-step process, taking the received signals and directly estimates the position of the object (often a transmitter). In direct positioning, the data received in all the sensors in a sensor network is first transmitted to a single location where the computation will be done.

Consider a signal, r_i, received at the i:th sensor:

$$r_i(t) = b_i a_i(\boldsymbol{p}) s(t - \tau_i(\boldsymbol{p}) - t_0) + \boldsymbol{n}_i(t) \qquad (2.1)$$

where b_i is a complex scalar describing the attenuation, a_i is the array response (see the section on Angle of Arrival in Two-Step Positioning), s is the signal waveform, τ_i is the delay from t_0 (the start time of the signal) and lastly the vector $\boldsymbol{n}_i$ is the noise. As can be seen, both the delay and the array response is dependent on $\boldsymbol{p}$, which is the position of the object (transmitter). This position is then determined (after some manipulation of the signals such as Fourier transforming them) by minimizing the cost function $Q(\boldsymbol{p})$:

$$Q(\boldsymbol{p}) = \sum_{i=1}^{L}\sum_{k=0}^{N-1}\left\|\bar{r}_i(k) - b_i a_i(\boldsymbol{p})\bar{s}(k)e^{-j\omega_k[\tau_i(\boldsymbol{p})+t_0]}\right\|^2 \quad (2.2)$$

where L is the total number of sensors, N is the number of discrete samples, $\bar{r}_i$ is the Fourier transform of r_i, $\bar{s}$ is the Fourier transform of s and $\|\bullet\|$ denotes the Frobenius norm [5].

This is in fact the least squares method. In the case of zero-mean Gaussian noise, however, this is the exact maximum likelihood estimate. The accuracy of DPD has proven to be better than the two-step positioning approach. This is, however, at the cost of high computational complexity due to the calculations of eigenvalues, amongst other things [6].

2.2 Two-Step Positioning

The first part of the Two-Step Positioning approach is to estimate position related parameters. The most common position related parameters are Received Signal Strength (RSS), Angle of Arrival (AOA), Time of Arrival (TOA) and Time Difference of Arrival (TDOA). Below, a node can either be a transmitter, receiver or simply an object to be located.

2.2.1 RSS

Since the amplitude (energy) of a signal reduces as it propagates through space (called path loss), the distance between a reference node and a target node can be estimated via the different strengths of the signals. However, due to several mechanisms such as diffraction, reflection and scattering, the relation between distance and signal power can be quite complicated. There are also other effects such as shadowing which causes errors in the estimation. These errors are often modeled as zero-mean Gaussian with a specific variance σ^2. How accurate a distance estimate can be is defined by the Cramer-Rao lower bound:

$$\sqrt{Var(\hat{d})} \geq \frac{\ln(10)\cdot\sigma\cdot d}{10n} \quad (2.3)$$

where $\hat{d}$ is an estimate of the distance d, σ is the square root of the variance of the error and n is the path loss exponent. As can be seen in (2.3), the higher the path loss exponent and the shorter distance between sensor and object, the better variance can theoretically be achieved.

2.2.2 AOA

The Angle of Arrival specifies the angle between a reference node and a target node, as can be seen in figure 3. The angle is often retrieved by having multiple antennas at the node receiving the signals. The difference in time of arrival for the different antennas then contains the angle information. The Cramer-Rao lower bound on the error (variance) can be calculated here as well. If the antennas are distributed equally along a line, the CRLB is:

$$\sqrt{Var(\hat{\alpha})} \geq \frac{\sqrt{3}c}{\sqrt{2}\pi\sqrt{SNR}\beta\sqrt{N_a(N_a^2-1)}l\cos(\alpha)} \quad (2.4)$$

where $\hat{\alpha}$ is the estimation of the Angle of Arrival α, the SNR is the signal to noise ratio for the sensor, β is the effective bandwidth, N_a is the number of antennas and l is the distance between two antennas. In (2.4) it is evident that the more antennas as well as higher SNR lead to a better theoretical variance.

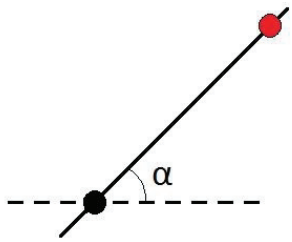


Figure 3. The Angle of Arrival, α

2.2.3 TOA

In Time of Arrival the time it takes for a signal to travel from a transmitter to a receiver (alternatively from a transmitter via a reflector back to the transmitter which also acts as a receiver) is measured and this time can then be converted to a distance (via the speed of light). As for RSS and AOA, TOA is also affected by noise. The noise is often modeled as zero-mean white Gaussian noise and the received signal at a sensor node can then be expressed as:

$$r(t) = s(t - \tau) + n(t) \tag{2.5}$$

where s is the transmitted signal from another node, τ is the Time of Arrival and $n(t)$ is the noise. With this model, the Cramer-Rao lower bound can be expressed as:

$$\sqrt{Var(\hat{\tau})} \geq \frac{1}{2\sqrt{2}\pi\sqrt{SNR}\beta} \tag{2.6}$$

where $\hat{\tau}$ is the estimation of τ (Time of Arrival), SNR is the signal to noise ratio in the sensor and β is the effective bandwidth. In (2.6) one can see that the larger the effective bandwidth as well as higher SNR, the better theoretically variance can be achieved.

2.2.4 TDOA

Time Difference of Arrival is basically an extension of Time of Arrival. There are two approaches for obtaining the TDOA. The first approach is to estimate the TOA for two signals between two different (synchronized) reference nodes and the target node. Then the TDOA estimate is simply the difference between the TOA estimates. The second approach is to cross-correlate two signals from two different reference nodes. The TDOA can then be calculated from the highest correlated value. The Cramer Rao lower bound is closely related to that of TOA.

The second part of the two-step positioning approach is to estimate the node position based on the estimates of the position related parameters. There are three main techniques for doing this, namely Mapping, Geometric and Statistical methods.

2.2.5 Mapping

This technique uses an existing database containing previous estimates of parameters. These estimates are then used to estimate the position of the node. The data in the database is often created by training a system before using it.

2.2.6 Geometric

The geometric methods use geometric relationships in order to calculate the position based on the estimated parameters. Depending on which position related parameters estimated, different relationships can be used.

When RSS or TOA is used, the distances between the nodes are known. Each distance, corresponding to the distance between one reference node and the target node, gives rise to a circle around the reference node where the target node can be located. In order to obtain a unique location of a target node in two dimensions, distances from three, linearly independent, reference nodes are needed. This is true if the objects can be located anywhere in the plane (which is the general case and therefore discussed here). But if the objects are only located in one half-plane, as is the specific case in this thesis, only two sensors are needed. More on this can be found in section 5.1.1. The three circles intersect in one point, which then is the location of the target node, see figure 4. This method is called trilateration. In three dimensions, four, linearly independent, nodes are needed to find the location. The location is now the intersect point of four spheres [7]. It is, of course, possible to find the location by using more reference nodes; this is then often called multilateration or simply lateration [8].

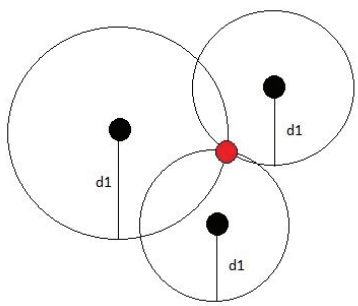


Figure 4. Estimating the node position (red dot) with three sensors (black dots) using trilateration in two dimensions

If AOA is used, the position of a node in 2 dimensions can be calculated using triangulation instead. The triangulation method uses the angles between two different reference nodes and the target node to calculate the position, see figure 5.

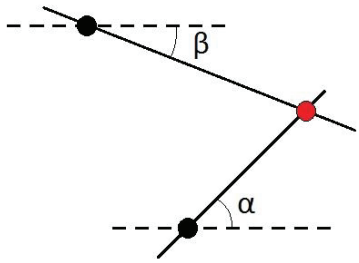


Figure 5. Estimating the node position (red dot) with two sensors (black dots) using triangulation

And lastly if TDOA is used, two distance differences can be obtained from two reference nodes with respect to a third one. Each distance difference gives rise to a hyperbola and the position of a target node in two dimensions is located where the two hyperbolas intersect, see figure 6.

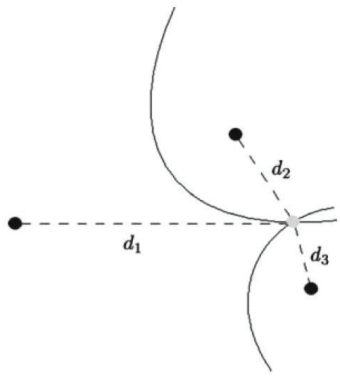


Figure 6. Estimating the node position (grey dot) with three sensors (black dots) using hyperbola intersection from TDOAs [9]

2.2.7 Statistical

Due to the fact that the estimates of the position related parameters are error prone (due to noisy measurements) statistical methods may be a better choice for estimating the position. For example, consider (geometric) trilateration in two dimensions. If the distances d1, d2 and d3 are wrongly estimated, the circles will not intersect at the same point and no explicit solution is found, see figure 7. Instead different methods based on the maximum likelihood as well as the least squares approaches can be used. More on statistical methods can be found in section 5.1.2. [4][9].

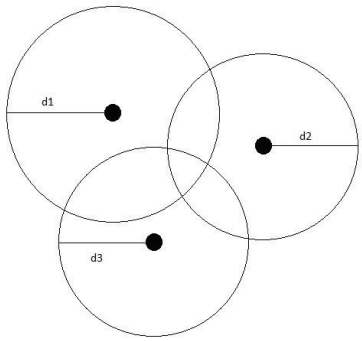


Figure 7. Estimating the node position with three sensors (black dots) using trilateration but with noisy measurements in two dimensions

3 Time Delay Estimation

Time delay estimation is, as explained in the background section (2), the first step in the two-step positioning scheme. This section will present the implementation of some algorithms as well as some results achieved.

3.1 Theories and Algorithms

Below, the three algorithms mentioned in the Problem Definition and Aim of the Thesis section are presented.

3.1.1 Peak Detection

This algorithm is the most simple of the Time Delay Estimations. When having a signal as in figure 7, the algorithm simply finds the positions of the two peaks, which then are the Time Delay Estimations. It does this by first finding all peaks (a peak is defined as a point which has a higher value than both its neighbors) of the signal. Then it finds all the peaks above a certain noise threshold, two in this case, see figure 8. The corresponding distances is then easy to extract from the input signal.

The reason for implementing this algorithm is its simplicity; due to the low complexity of the algorithm, the computational power needed is low.

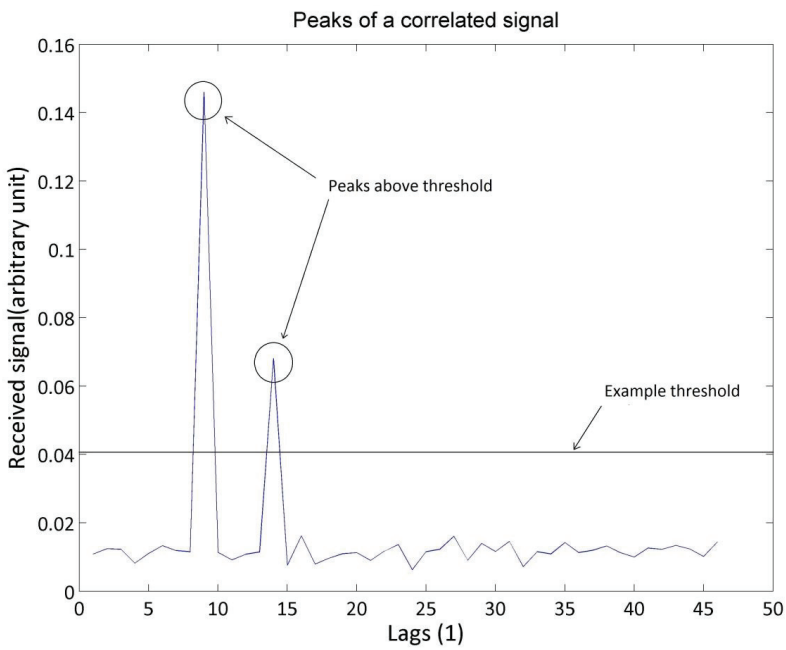


Figure 8. Peaks of an input signal with a noise threshold in order to find the signal peaks

3.1.2 Covariance

The covariance method for determining the Time Delay Estimation is a bit more complex than the Peak Detection method. It uses the covariance:

$$C(X,Y) = E[(X - \mu_x) \cdot (Y - \mu_y)] \tag{3.1}$$

where X is the received signal and Y is a reference signal, seen in figure 9, and $E[\cdot]$ is the expected value. The covariance is a measure of how dependent two variables are on each other. If they are independent the covariance is zero. If they are dependent, however, the covariance is positive [10]. If the reference signal is swept with respect to the received signal and the covariance is calculated at each step, the result is a new vector, see figure 10. This is called a cross-covariance. The peaks in the resulting vector are located where the delay between the reference signal and the received signal is such that the vectors are the most dependent. Since the time delay is known for the reference signal, the Time Delay Estimations can be calculated as the addition of the positions of the peaks and the reference signal time delay. The positions are found in the same way as in Peak Detection.

The reason for implementing this algorithm is mainly its robustness. It has a much higher tolerance for noise than the Peak Detection method.

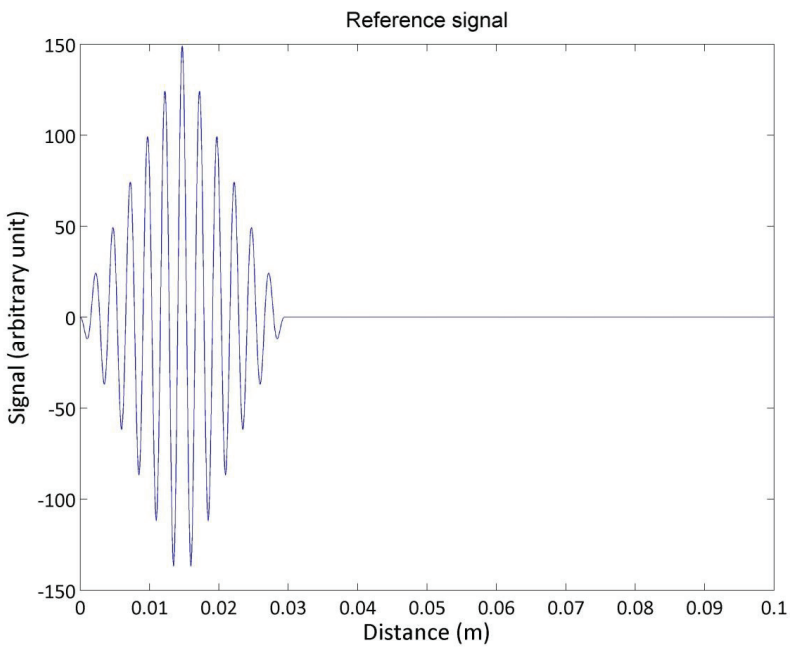


Figure 9. A reference signal

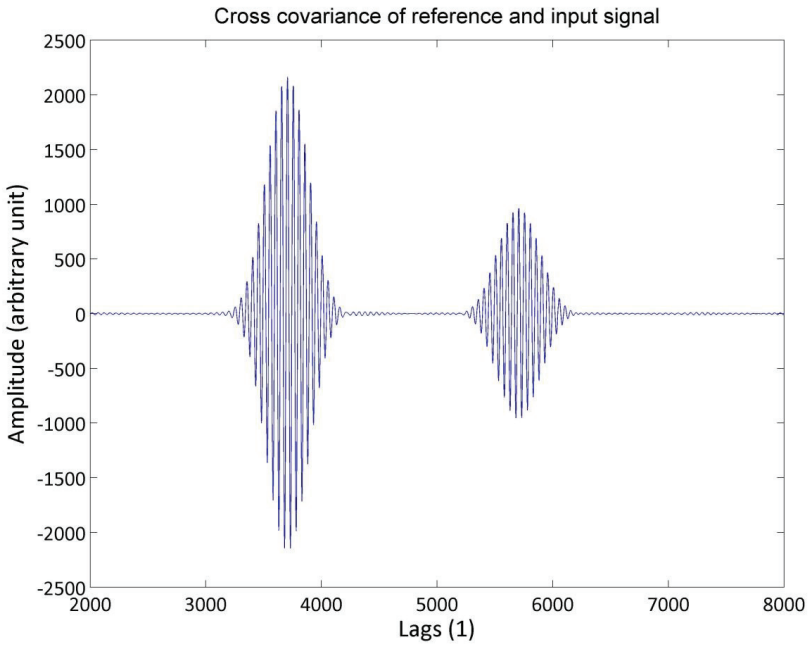


Figure 10. Cross covariance of the reference signal in figure 9 and the input signal in figure 7

3.1.3 Least Squares

This method is the most complex of the three. It models the received signal, $r(t)$ as

$$r(t) = \sum_{l=1}^{L} \alpha_l s(t - \tau_l) + n(t) \tag{3.2}$$

where $s(t)$ is the reference signal, L is the number of reflections, α_l is the amplitudes for the different reflections, τ_l is the delays for the different reflections and $n(t)$ is the noise. The noise is modeled zero-mean white Gaussian noise.

The signals in the model (received, reference and noise) are Fourier transformed and then the amplitudes and delays are estimated by minimizing the non-linear least squares criterion:

$$C(\alpha_l, \omega_l) = ||Y - \alpha_l S a(\omega_l)||^2 \tag{3.3}$$

where S is the Fourier transformed reference signal, $a(\omega_l)$ is the Fourier transform of a time delay and Y is the Fourier transform of the received signal. Minimizing C with respect to ω_l and α_l gives the estimates:

$$\widehat{\omega}_l = \arg\max |a^H(\omega_l) \cdot (SY)|^2 \tag{3.4}$$

and

$$\hat{\alpha}_l = \frac{a^H(\omega_l) \cdot (SY)}{||S||^2}\Big|_{\omega_l=\hat{\omega}_l} \tag{3.5}$$

where $a^H(\omega_l)$ is the Fourier transform. It is the $\hat{\omega}_l$ that is the most interesting since it can be converted to a Time Delay Estimation via:

$$\tau_l = \omega_l N T_s \tag{3.6}$$

where N is the number of bins and T_s is the sampling period. The maxima of $|a^H(\omega_l) \cdot (SY_l)|^2$ is found in the same way as the Peak Detection algorithm.

The reason for implementing this algorithm is actually the higher complexity; maybe there is a decent enough tradeoff in accuracy and stability to compensate for this complexity. [11]

3.2 Test Setup

To be able to evaluate the three algorithms for time delay estimation the following setup was used.

3.2.1 Signal Generation

Analog signals

One sensor is placed in origo and the system parameters are set so that a reflection from an object located 0.3 m away from the sensor at an angle where the sensor's directivity is at maximum (angle=90 degrees), has a SNR of approximately 2.2 (or 3.4dB). This value is based on a signal generation algorithm imitating a real system. The SNR is calculated by dividing the rms value of the reflection without noise and the rms value of the noise:

$$SNR = \frac{\sqrt{\frac{\sum_{i=1}^{N} s_i}{n}}}{\sqrt{\frac{\sum_{i=1}^{N} n_i}{n}}} \tag{3.7}$$

where n is the length of a reflection and s_i and n_i are the signal and noise, respectively, at the i:th point in the correlated signal vector.

The signal generated has a resolution as good as 50 micrometers and can therefore be regarded as almost analog. A generated analog signal can be seen in figure 7.

Digital signals

The digital signals are generated with a more advanced script where the signals generated bear much more resemblance to the reality. More noise sources are present and no SNRs are calculated. The

signals have a resolution of 0.5 mm and cannot be regarded as analog and are therefore called digital signals. A generated digital signal can be seen in figure 11.

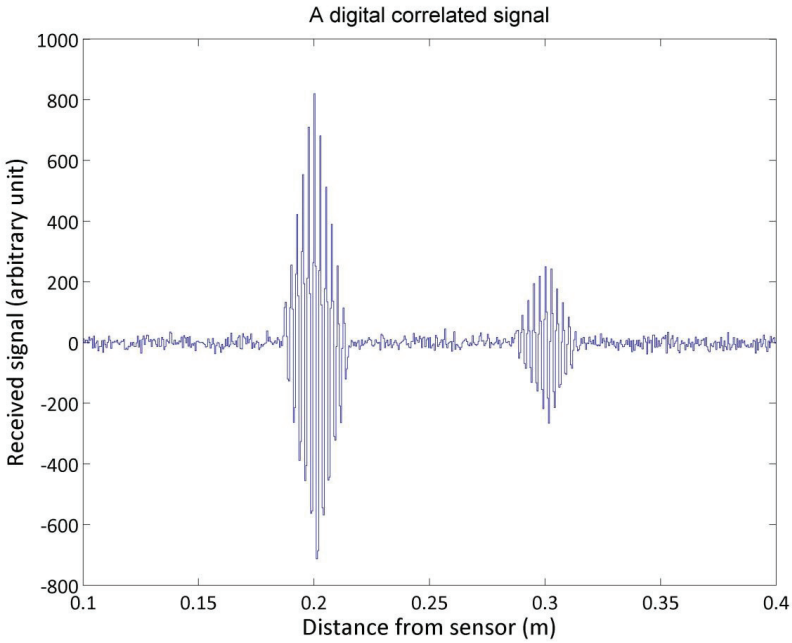


Figure 11. A correlated digital signal with reflections 0.2 m and 0.3 m away

3.2.2 Test Execution

In the test, 10 000 simulations are conducted.

In each simulation, one object is randomly placed within the Area. The SNR of the reflection from the object is then in the range -20 dB to 60 dB. The distance to the object is estimated using the three algorithms under evaluation, namely, Peak Detection, Covariance and Least Squares. The estimation error for each method is saved as well as the SNR of the generated signal vector.

Once the simulations are finished, the estimation errors for each of the three algorithms are sorted and plotted, in the same plot, as a cumulative distribution function. The plot's x-axis is limited to 0.2 mm.

The estimation error as a function of SNR is also plotted, as a scatter plot, for the three algorithms. The number of times the estimation fails (due to not finding the reflection) is recorded for the N simulations, as well as the number of estimation errors larger than 0.2 mm. The average runtimes for the algorithms are also recorded and lastly the variance for the estimation errors is calculated.

After these simulations, 1000 new simulations are conducted with digital signals. The reason for the relatively few simulations is that the generation of the more reality based signals is time-consuming.

The setup is the same but no SNR plots are done. The reason for this is the difficulty in actually defining the SNR due to the many different noise sources in the generation of the signals.

3.3 Results

The results are separated into three categories, namely stability, accuracy and computational complexity. All the results for the Time Delay Estimation are presented below.

3.3.1 Analog

Stability

The stability is mainly based on if a reflection is found or not and below is a table presenting this for the three different algorithms.

Algorithm	Percentage not found
Peak Detection	21.1%
Covariance	0.0%
Least Squares	0.0%

Table 1. Percentage of the reflections not found for the algorithms with analog signals

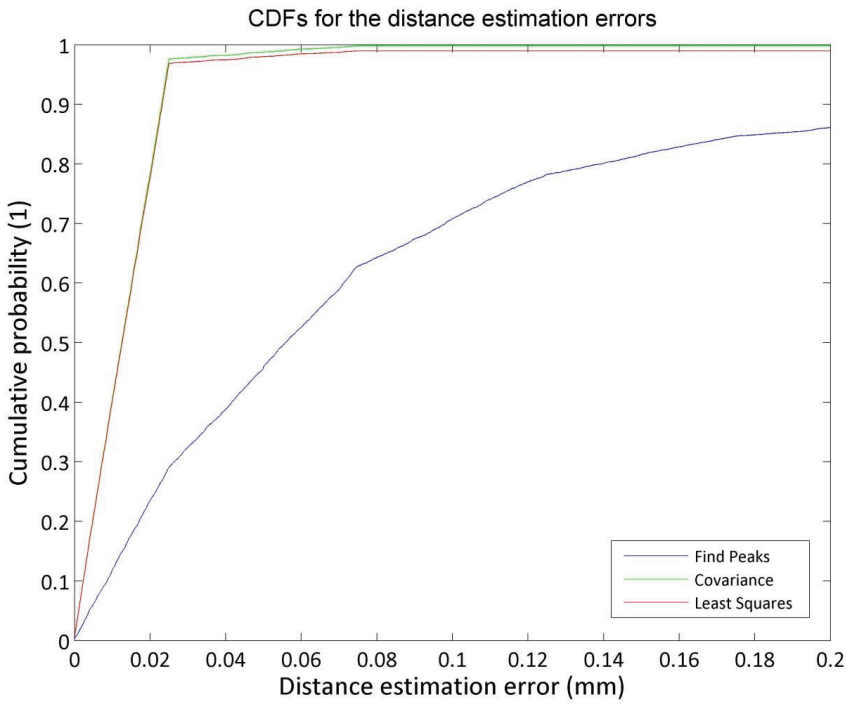
The percentage of large errors can also be considered as a parameter for stability. The table below displays these large errors for the algorithms. An error is considered large if it is larger than 0.2 mm.

Algorithm	Percentage large errors
Peak Detection	13.9%
Covariance	0.2%
Least Squares	1.0%

Table 2. Percentage of large (>0.2 mm) errors for the algorithms with analog signals

Accuracy

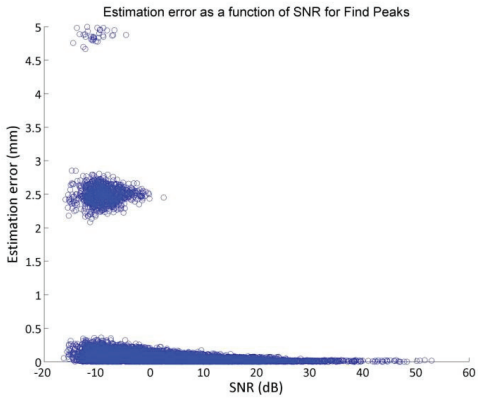
To see the accuracy of the algorithms a cumulative distribution function plot of the errors has been done and can be seen in plot 1.



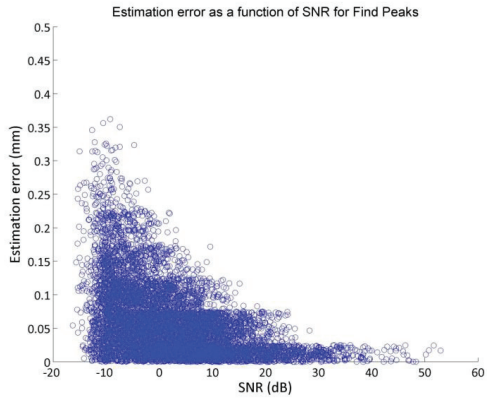
Plot 1. Cumulative distribution functions of the errors for the algorithms with analog signals

Since the signal strength of the reflections is weaker, and hence the SNR is worse, for objects far away, it is also interesting to investigate the error depending on SNR. Six SNR plots are presented next. The plots on the left-hand side display errors up to 5 mm while the plots on the right-hand side are zoom-ins of those on the left-hand side, displaying only errors up to 0.5 mm.

The two SNR plots for Peak Detection:

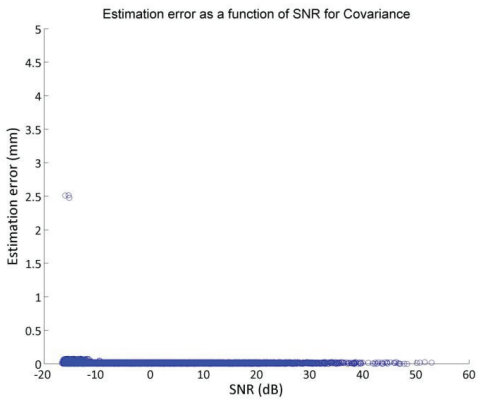


Plot 2. Estimation error as a function of SNR for Peak Detection, high-valued y-axis

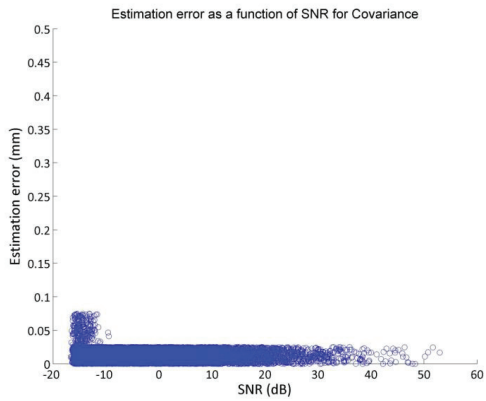


Plot 3. Estimation error as a function of SNR for Peak Detection, low-valued y-axis

The two SNR plots for Covariance:

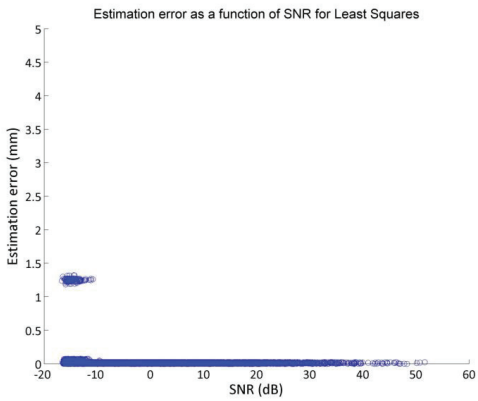


Plot 4. Estimation error as a function of SNR for Covariance, high-valued y-axis

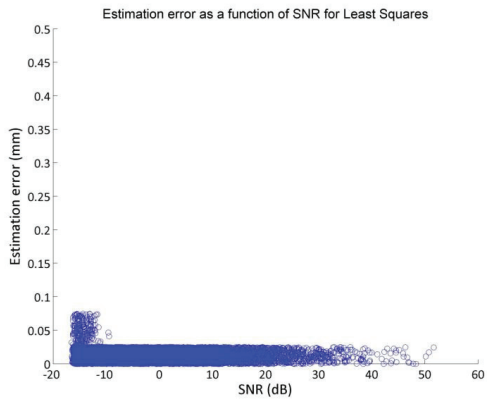


Plot 5. Estimation error as a function of SNR for Covariance, low-valued y-axis

The two SNR plots for Least Squares:



Plot 6. Estimation error as a function of SNR for Least Squares, high-valued y-axis



Plots 7. Estimation error as a function of SNR for Least Squares, low-valued y-axis

Computational Complexity

The third parameter to investigate is the computational complexity, i.e. how time-consuming these algorithms are. Below is a table showing the average runtime for each algorithm. Note that these runtimes are for one sensor only.

Algorithm	Average runtime
Peak Detection	18.6 ms
Covariance	23.4 ms
Least Squares	21.2 ms

Table 3. Average runtime for the algorithms with analog signals

3.3.2 Digital

The results are presented in the same way as for the analog signals, with the exception that no SNR plots have been made (due to the difficulties to actually define the SNR in the presence of a large number of different noise sources).

Stability

The corresponding table for percentage of reflections not found with digital signals is found below.

Algorithm	Percentage not found
Peak Detection	34.3%
Covariance	0.0%
Least Squares	1.7%

Table 4. Percentage of the reflections not found for the algorithms with digital signals

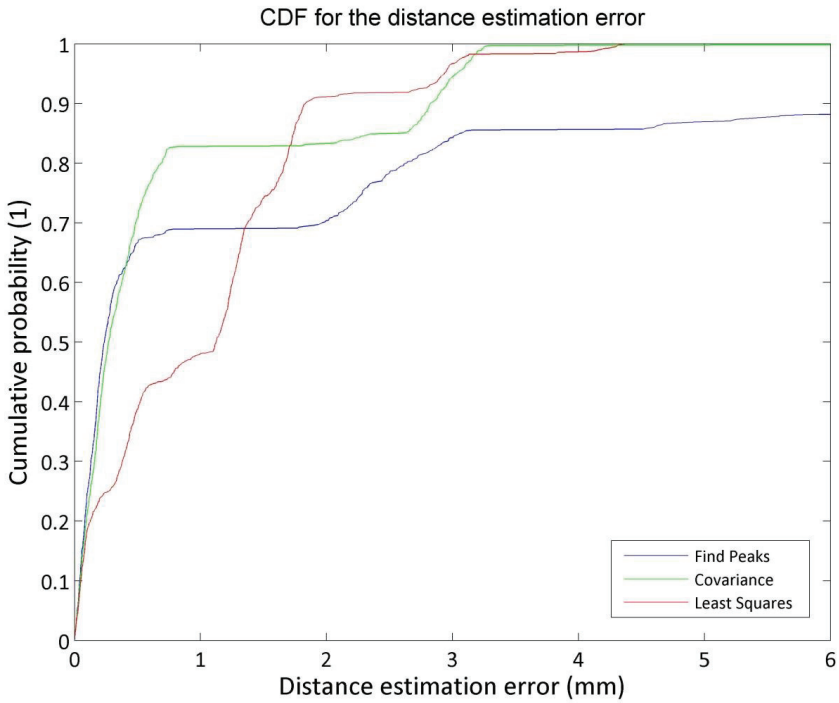
An error is now considered large if it is larger than 3.0 mm.

Algorithm	Percentage large errors
Peak Detection	16.9%
Covariance	5.40%
Least Squares	3.76%

Table 5. Percentage of large (>1.0 mm) errors for the algorithms with digital signals

Accuracy

In plot 8, the cumulative distribution functions for the errors are presented.



Plot 8. Cumulative distribution functions of the errors for the algorithms with digital signals

Computational Complexity

The average runtimes will be significantly shorter due to the fact that the digital signals have a lower resolution and hence fewer data points.

Algorithm	Average runtime
Peak Detection	4.16 ms
Covariance	6.59 ms
Least Squares	6.60 ms

Table 6. Average runtime for the algorithms with digital signals

3.4 Discussion and Conclusions

3.4.1 Analog

Stability

As can be seen in table 1, the by far most unstable algorithm is Peak Detection, whereas both Covariance and Least Squares are perfectly stable with these analog signals. This is not surprising since Peak Detection only searches for the highest data point and hence if the noise is large enough the algorithm cannot distinguish the signal from the noise. Since the other two algorithms considers not just the highest data point, but the input signal as a whole and, in a way, compares it with a reference signal, a correct distance estimation can be made even if the signal is weaker than the noise.

The fact that Peak Detection only considers the highest data point makes its estimation accuracy more heavily influenced by noise in comparison with the other two algorithms, as can be seen in table 2. Consider some noise making a data point have a higher value than the true peak, this data point is selected and a distance estimation error occurs. If this happens for the other two algorithms however, it does not have a significant impact due to that these algorithms consider the whole input signal, and not just the highest data point.

Accuracy

It is already stated above that the Covariance and the Least Squares algorithms have better accuracy than Peak Detection, and this is confirmed in plot 1. Here, one can see that with these analog input signals, the Covariance and Least Squares algorithms basically estimates the distance without error excluding a few outliers. These outliers are explained in plots 4 and 6. In plot 4, one can clearly see that the outliers have an error of 2.5 mm which is the length of one period in the correlated input signal. This of course only happens if the SNR is low. In plot 5, the outliers have an error of 1.25 mm, which corresponds to half a period in the correlated input signal. The reason that the Least Squares algorithm have errors that is 1.25 mm (as opposed to a full period) is when the algorithm Fourier transforms the signal, only positive values (frequencies) are produced and hence the algorithm does not distinguish between a signal peak and a signal trough. See figure 12 to more easily understand the reason of these outliers.

The fact that the algorithms have outliers corresponding to the periods of the input signal is perhaps even more evident in plot 2. In the zoom-ins (i.e. plots 3, 5 and 7) the estimation errors seem to appear in “steps”. These steps are around 0.05 mm in size and exactly correspond to the resolution of the input signal. This effect arises due to the algorithms picking neighboring data points (of the true value) as estimations.

Once again it is clear that the Covariance and Least Squares algorithms by far outperform the Peak Detection algorithm.

Computational Complexity

The average runtimes are of course a very important factor when evaluating the algorithms. From table 3 one apparent conclusion can be made; there is actually no big difference in the runtimes for

the algorithms. Peak Detection is, as expected due to its simplicity, the best algorithm when it comes to computational complexity, but not by a large margin. Of course, to be said is that these runtimes are implementation dependent and may not be the best possible for the algorithms. With more time and effort, the algorithms could be made to be more efficient, for example by implementing them in native C instead of in Matlab. The reason that the Covariance algorithm has a bit longer runtime than the Least Squares algorithm is that the actual covariance operation is of $O(n^2)$ whereas the FFT operations (three in total) in the Least Squares algorithm is of $O(n*log(n))$. The rest of the algorithms (which basically is finding the peaks in the resulting vectors) is almost the same, hence the small difference.

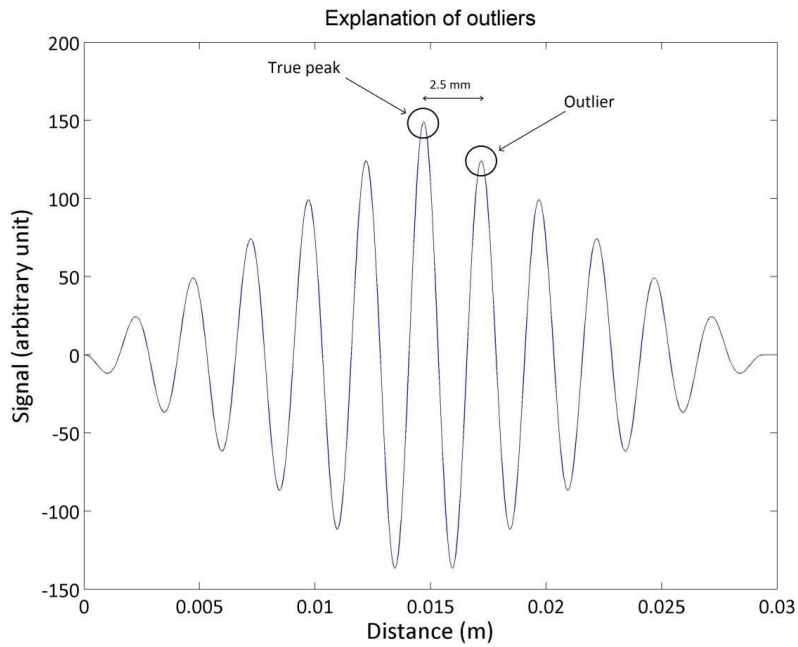


Figure 12. A correlated signal with its true peak and the side peaks being outliers

3.4.2 Digital

Stability

Just as with analog signals, the Peak Detection algorithm is by far the worst when it comes to stability, as seen in table 4. With digital signals the algorithm is even worse with over 30% reflections not found. One big difference when it comes to percentage reflections not found is that the Covariance algorithm still finds all the reflections while the Least Squares algorithm does not. The reason for this is that for noisier signals that aren't of Gaussian nature, the Least Squares algorithm has more difficulties finding the correct peaks in the frequency vector, (3.4).

The percentage large errors are quite small for both Covariance and Least Squares but higher for Peak Detection, which can be seen in table 5.

Accuracy

In plot 8, plateaus for the cumulative distribution functions of the errors can be seen for all three algorithms. The plateaus represent a range where there are practically no errors. This is due to the period of the correlated input signal; the same phenomenon that could be seen in the scatter plots for analog signals (i.e. the outliers). This is especially apparent when comparing Least Squares with the other two algorithms. Least Squares have more plateaus since the errors can be half a period as well as a full period.

Computational Complexity

As for analog signals, the three algorithms are quite similar when it comes to runtimes, see table 6. The runtimes for the digital signals are much shorter than the runtimes when having analog signal. This is not very surprising since the resolution is a factor 10 poorer for the digital signals and hence fewer data points needs to be processed by the algorithms.

3.4.3 Conclusion

Considering all the observations for both analog and digital signals the best algorithm is either Covariance or Least Squares. When running in an application the most important quality is the stability, assuming reasonable computational complexity. Therefore, the time delay estimation algorithm to be implemented in the Android application is Covariance since it found all reflections and the runtime was only a little slower than the runtime for Least Squares. Also, the Covariance algorithm is quite easy to implement in Java Android.

4 Sensor Distribution Analysis

Before the thesis will proceed to the position estimation part, some sensor distribution analysis will be performed. The reason for this is to first find out how to place the sensors as well as how many in the tests on position estimation algorithms, before actually performing them.

4.1 Sensor Placement

This section evaluates the position error with respect to sensor placement.

4.1.1 Test Setup

Two Dimensions:

In the test, 10 000 simulations are conducted.

In each simulation, one object is randomly placed within the Area. Two sensors are placed on the x-axis with the distance between them being varied from 0.002 m to 0.1 m with a step-size of 0.002 m. The sensors are placed symmetrical around the y-axis (for example, the sensors can be placed at (-0.04 0) and (0.04 0). The distance errors for both sensors are constant at 1 mm and the position error for each sensor placement is calculated (using the algorithm presented in section 5.1.3) and saved.

After the simulations, the average position error is calculated for all the sensor placements and then plotted against the distance between the sensors.

Three Dimensions:

In the test, 10 000 simulations are conducted.

In each simulation, one object is randomly placed within the Volume. Three sensors are placed in the xy-plane creating a triangle with the sensors as vertices. The area of this triangle is varied from $0.013cm^2$ to $32.5cm^2$. The sensors are placed in such a way that the triangle becomes equilateral and has its centroid in origo, (0,0). The distance errors for all three sensors are constant at 1 mm and the position error for each sensor placement is calculated (using the algorithm presented in section 5.1.3) and saved.

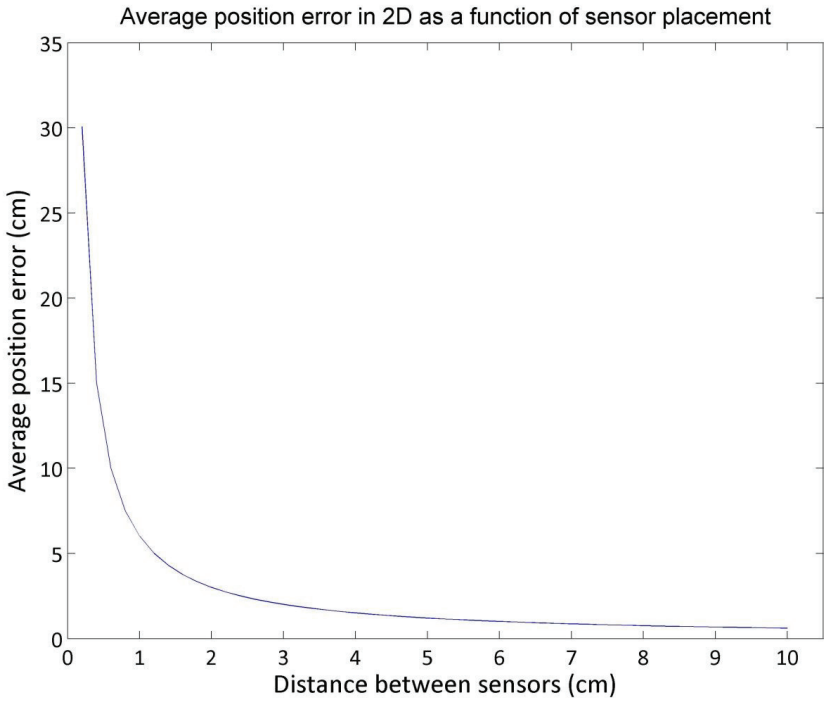
After the simulations, the average position error is calculated for all the sensor placements and then plotted against the area for the triangle with the sensors as vertices.

4.1.2 Results

The results are first presented as average position error as a function of either distance or area, depending on dimension. Then four figures are displayed to easier visualize the results.

Two Dimensions:

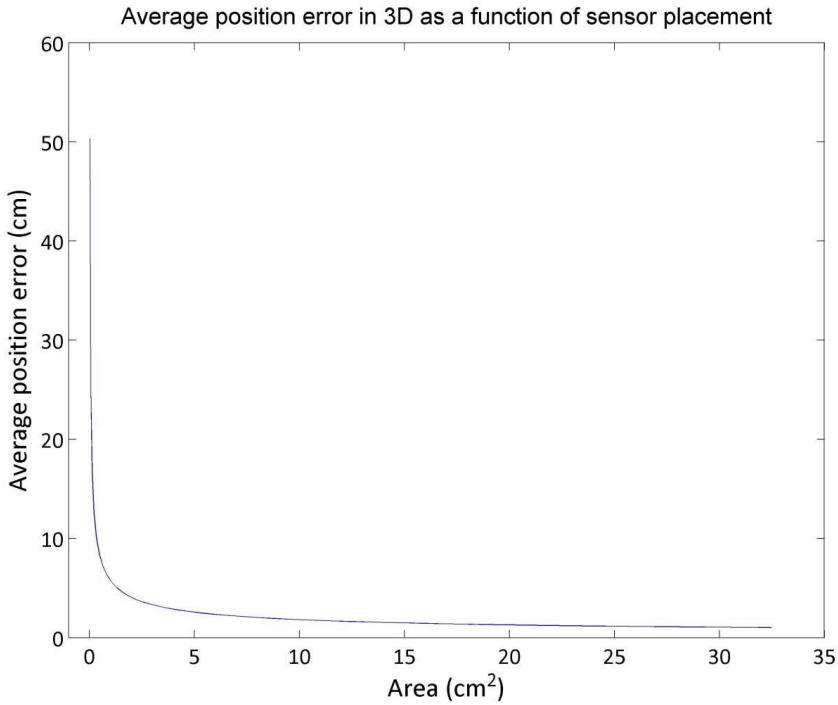
Plot 9 presents the results of the two dimensional simulations in the form of average position error as a function of distance between the two sensors.



Plot 9. Average position error as a function of distance between two sensors in two dimensions, with a constant distance error of 1.0 mm for both sensors

Three Dimensions:

In plot 10, the results of the three dimensional simulations are presented in the form of average position error as a function of the triangle area between the three sensors.



Plot 10. Average position error as a function of area between three sensors in three dimensions, with a constant distance error of 1.0 mm for all three sensors

The two figures below shows the difference in position error depending how the sensors are placed.

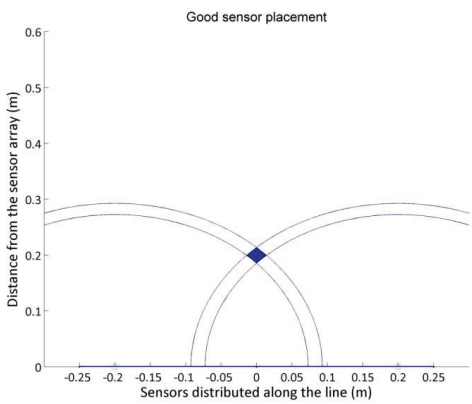


Figure 13. Position error for well-placed sensors

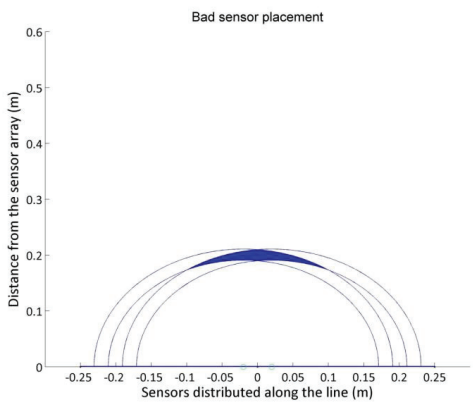


Figure 14. Position error for ill-placed sensors

The next two figures also present the difference in position error, but these are based on actual probabilities for where the object might be. Red color represents a low probability that the object is found there while blue color represent a high probability. The two small green circles located along the x-axis are the sensors.

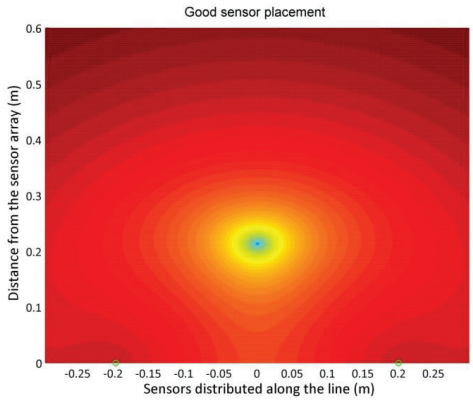


Figure 15. Position error probability for well-placed sensors

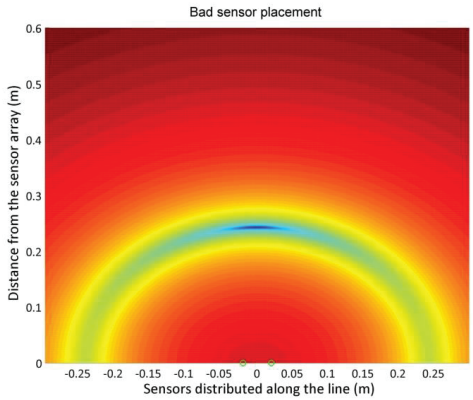


Figure 16. Position error probability for ill-placed sensors

4.1.3 Discussion and Conclusions

One thing to first note before evaluating the position placement is that the values on the y-axis are not especially significant. The test is done in such a way that the overall behavior may be analyzed but the actual values do not mean anything. The reason for this is that the position errors are calculated using an error algorithm (presented in section 5.1.3) which takes constant values of the distance errors as input, i.e. it is not a true estimation algorithm.

Two Dimensions:

It is evident from plot 9 that the average position error is greatly reduced when the distance between the sensors is as large as possible. However, around 4 or 5 cm, the difference in accuracy becomes substantially less significant. The conclusion is then simply to have the sensors as far apart as possible.

Three Dimensions:

It is equally evident from plot 10 that the position error is smaller when the sensors are further apart (i.e. have a larger area between them) and thus the conclusion is the same as in the two-dimensional case; have the sensors as far apart as possible.

When looking at figures 12 to 15, one can see that the position error is affected differently in different directions. So for an object positioned in front of the sensors, the depth resolution is still quite good for sensors placed close to each other but the horizontal resolution is very bad.

4.2 Sensor Density

This section evaluates the position error with respect to the number of sensors placed.

4.2.1 Test Setup

Two Dimensions:

1000 simulations are conducted.

In each simulation, one object is randomly placed within the Area. The number of sensors is changed from 2 to 30 and they are placed on the x-axis, equidistant from one another with the two outermost sensors at x=-0.05m and x=0.05m. The distance estimation noise is of zero mean Gaussian nature with a standard deviation of 1 mm. The position is estimated using a brute force maximum likelihood algorithm in two dimensions, which will be presented in section 5.1.2. The estimated position error as well as the runtime are calculated and saved.

After the simulations, the average position errors as well as the average runtime for the different number of sensors are calculated and plotted versus the number of sensors. The reason that only 1000 simulations are performed (as opposed to 10 000 in Sensor Placement) is due to the fact that the brute force algorithm's execution time is quite long.

Three Dimensions:

1000 simulations are conducted.

In each simulation, one object is randomly placed within the Volume. The number of sensors is changed from 3 to 10. The sensors are placed in circles in the xy-plane; doing this is an efficient way of placing the sensors and still keep the number of linearly dependent sensors to a small amount (which is quite important, as will be seen further ahead in the thesis). The distance estimation noise is of zero mean Gaussian nature with a standard deviation of 1 mm. The position is estimated using a brute force maximum likelihood algorithm in three dimensions, which will be presented in section 5.1.2. The estimated position error as well as the runtime are calculated and saved.

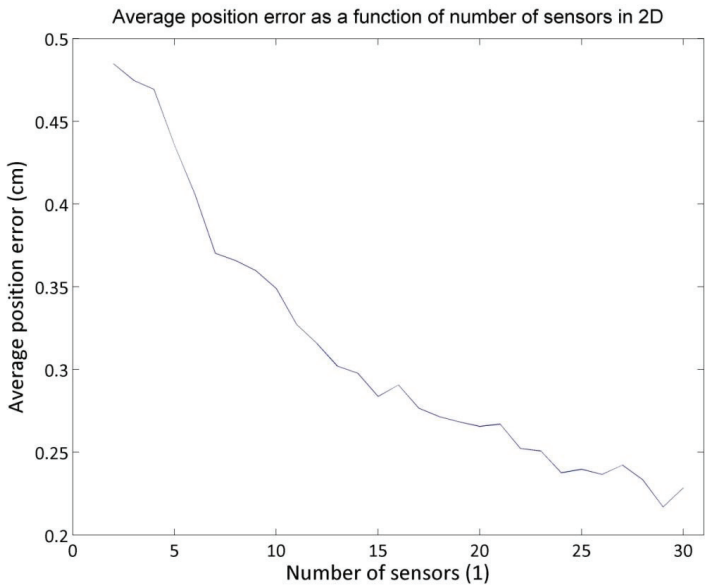
After the simulations, just as in the two dimensional case, the average position errors as well as the average runtime for the different number of sensors are calculated and plotted versus the number of sensors.

4.2.2 Results

The results are displayed in the form of two plots for both two dimensions and three dimensions.

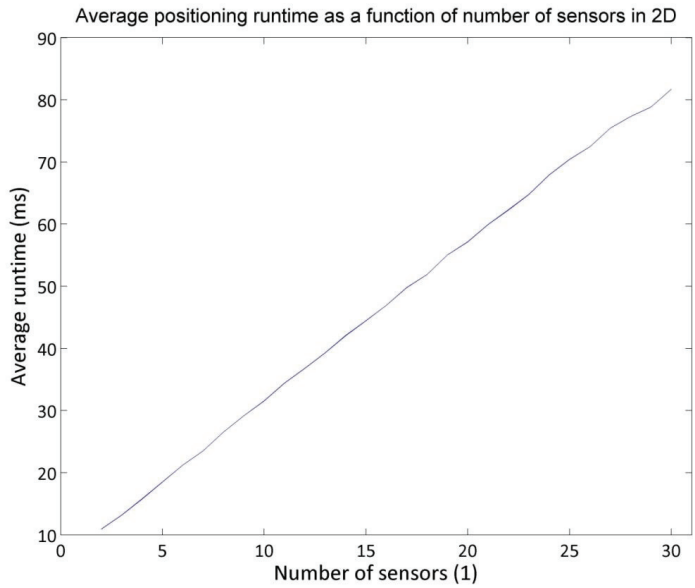
Two Dimensions

The first plot shows the average position error as a function of number of sensors along the x-axis.



Plot 11. Average position error as a function of number of sensors in two dimensions

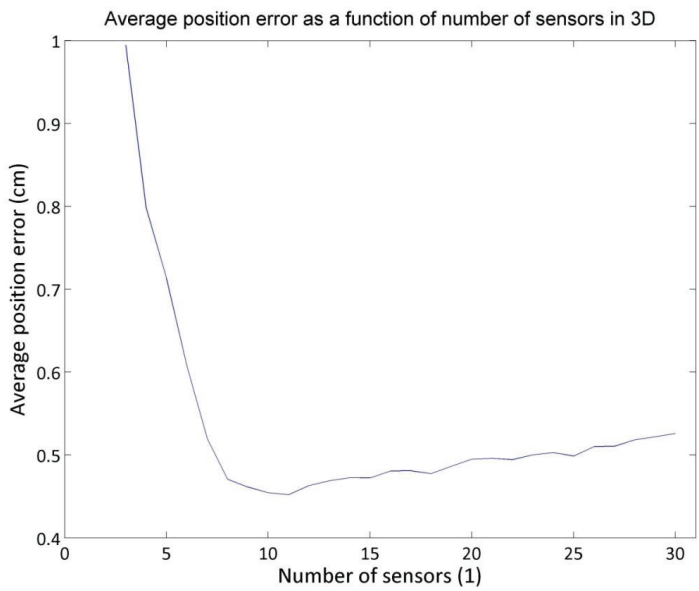
The second plot shows the average runtime as a function of number of sensors.



Plot 12. Average runtime as a function of number of sensors in two dimensions

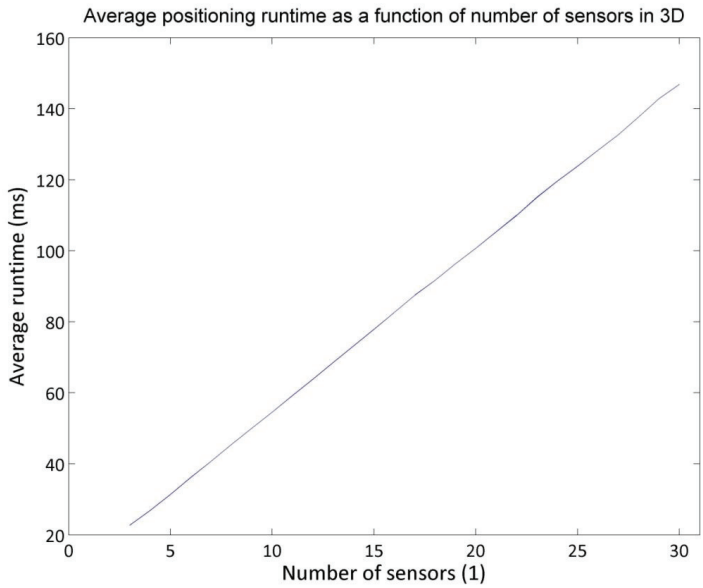
Three Dimensions:

The first plot shows the average position error as a function of number of sensors distributed in circles in the xy-plane.



Plot 13. Average position error as a function of number of sensors in three dimensions

The second plot shows the average runtime as a function of number of sensors.



Plot 14. Average runtime as a function of number of sensors in three dimensions

4.2.3 Discussion and Conclusions

Just as in section 4.1, this test is done in a way where the overall behavior can be analyzed and interpreted but the actual values are of lesser importance.

Two Dimensions:

From plot 11, one can deduct that the position error can be reduced by having more sensors. Also, from plot 12, the time complexity is of *O(n)*, where n is the number of sensors. However, oftentimes the number of sensors is limited by other parameters such as space and cost. For example, the number of sensors one can have in a phone may be limited and therefore a reasonable tradeoff has to be made.

Three Dimensions:

When the number of sensors ranges from 3 to around 10, the same behavior as in the two-dimensional case can be observed in plot 13. However, when more sensors are added the accuracy actually decreases. The reason for this is that the number of almost linearly dependent sensors increases and as an effect decreases the accuracy. More on linearly dependent / independent sensors can be found in section 5.1.1.

5 Localization for One Object

This section presents implementations of different localization algorithms, results achieved and some error analysis.

5.1 Theories and Algorithms

As described in the background, different localization methods exist. First the implementation of algorithms using the geometrical approach is presented, then implementation of algorithms using the statistical approach is presented and lastly, the theory behind the error analysis used in this Thesis is presented.

5.1.1 Geometrical

Below, the algorithms using the geometrical approach are presented.

5.1.1.1 Single Geometric Trilateration

Two Dimensions:

Since the sensors are placed along the x-axis and the object has to be located at y > 0, two sensors are needed to find the position of the object (as opposed to three sensors when there are no restrictions of the object's position in the plane). The goal is to find the intersection between the two half-circles generated by the distance formulas for the two sensors in two dimensions:

$$d_1^2 = (x - x_1)^2 + (y - y_1)^2 \quad (5.1)$$

$$d_2^2 = (x - x_2)^2 + (y - y_2)^2 \quad (5.2)$$

See figure 17 below for an explanation of the variables and parameters used.

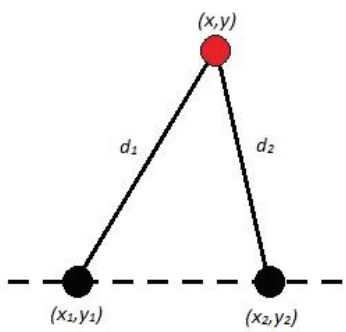


Figure17. Trilateration with two sensors (black dots) and one object (red dot) in two dimensions

Then the two distances are subtracted:

$$d_1^2 - d_2^2 = 2(x_2 - x_1)x + 2(y_2 - y_1)y + x_1^2 + y_1^2 - x_2^2 - y_2^2 \quad (5.3)$$

Since $y_1 = 0$ and $y_2 = 0$, (5.3) simplifies to:

$$d_1^2 - d_2^2 = 2(x_2 - x_1)x + x_1^2 - x_2^2 \quad (5.4)$$

Now x can be extracted as:

$$x = \frac{d_1^2 - d_2^2 - x_1^2 + x_2^2}{2(x_2 - x_1)} \quad (5.5)$$

and y as:

$$y = \pm\sqrt{d_1^2 - (x - x_1)^2} \quad (5.6)$$

Since the object only can be located at $y > 0$ the minus sign above can be ignored.

Three Dimensions:

Since the sensors are placed in the xy-plane and the object has to be located at z > 0, three sensors are needed to find the position of the object (as opposed to four sensors when there are no restrictions on the object's position in the room). The goal is to find the intersection between the three half-spheres generated by the distance formulas for the three sensors in three dimensions:

$$d_1^2 = (x - x_1)^2 + (y - y_1)^2 + (z - z_1)^2 \quad (5.7)$$

$$d_2^2 = (x - x_2)^2 + (y - y_2)^2 + (z - z_2)^2 \quad (5.8)$$

$$d_1^2 = (x - x_3)^2 + (y - y_3)^2 + (z - z_3)^2 \quad (5.9)$$

See figure 18 below for an explanation of the variables and parameters used.

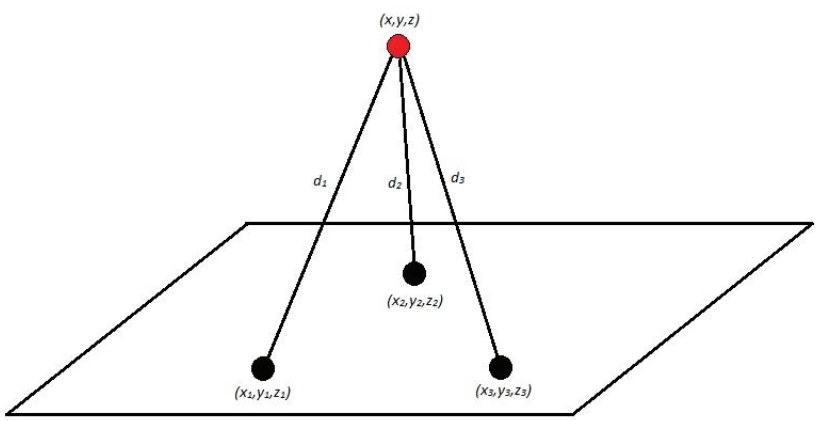


Figure 18. Trilateration with three sensors (black dots) and one object (red dot) in three dimensions

Then the first two equations are subtracted by the third:

$$d_1^2 - d_3^2 = 2(x_3 - x_1)x + 2(y_3 - y_1)y + 2(z_3 - z_1)z + x_1^2 + y_1^2 + z_1^2 - x_3^2 - y_3^2 - z_3^2 \quad (5.10)$$

$$d_2^2 - d_3^2 = 2(x_3 - x_2)x + 2(y_3 - y_2)y + 2(z_3 - z_2)z + x_2^2 + y_2^2 + z_2^2 - x_3^2 - y_3^2 - z_3^2 \quad (5.11)$$

Since $z_1 = 0$, $z_2 = 0$ and $z_3 = 0$, (5.10) and (5.11) simplify to:

$$d_1^2 - d_3^2 = 2(x_3 - x_1)x + 2(y_3 - y_1)y + x_1^2 + y_1^2 - x_3^2 - y_3^2 \quad (5.12)$$

$$d_2^2 - d_3^2 = 2(x_3 - x_2)x + 2(y_3 - y_2)y + x_2^2 + y_2^2 - x_3^2 - y_3^2 \quad (5.13)$$

These equations give rise to a linear system:

$$2(x_3 - x_1)x + 2(y_3 - y_1)y = d_1^2 - d_3^2 - x_1^2 - y_1^2 + x_3^2 + y_3^2 \quad (5.14)$$

$$2(x_3 - x_2)x + 2(y_3 - y_2)y = d_2^2 - d_3^2 - x_2^2 - y_2^2 + x_3^2 + y_3^2 \quad (5.15)$$

This system can be solved using gauss elimination and the solution is:

$$x = \frac{d_2^2 - d_3^2 - x_2^2 - y_2^2 + x_3^2 + y_3^2 - \frac{y_3 - y_2}{y_3 - y_1} \cdot (d_1^2 - d_3^2 - x_1^2 - y_1^2 + x_3^2 + y_3^2)}{2(x_3 - x_2) - \frac{y_3 - y_2}{(y_3 - y_1)} \cdot 2(x_3 - x_1)} \quad (5.16)$$

$$y = \frac{d_1^2 - d_3^2 - x_1^2 - y_1^2 + x_3^2 + y_3^2 - 2(x_3 - x_1)x}{2(y_3 - y_1)} \quad (5.17)$$

$$z = \pm\sqrt{d_1^2 - (x - x_1)^2 - (y - y_1)^2} \quad (5.18)$$

Since the object only can be located at $z > 0$, the minus sign before the square root in (5.18) can be ignored. Note here that if the sensors are placed in a line in the xy-plane (i.e. they are linearly dependent) there is no solution to the linear system above. Also, if sensors one and three are located at the same y-coordinate, the gauss elimination has to be conducted in a different order (but of course this is just as easy as above).

5.1.1.2 Centroid Trilateration

This algorithm is an extension of the "Single Geometric Trilateration for 1 Object".

Two Dimensions:

Now, the number of sensors is not restricted to two, but can be any number larger or equal to two. Firstly the maximum number of possible intersections, N, is calculated as

$$N = \binom{n}{2} \tag{5.19}$$

where n is the number of sensors and the number two represents the dimension. The reason that this is the maximum number of intersections is that this is the number of combinations of any two sensors (which give rise to an intersection) out of all n sensors. All the intersections are calculated for all the combinations of sensors. These intersections constitute a polygon with N vertices. And lastly, the centroid of this polygon is taken as the position estimate.

Three Dimensions:

In three dimensions, the number of sensors can be any number larger or equal to three. Firstly, similar to the two dimensional case, the maximum number of possible intersections, N, are calculated as

$$N = \binom{n}{3} \tag{5.20}$$

And similar to the two dimensional case, n is the number of sensors and the number three represents the dimension. This is the number of combinations of any three sensors (which give rise to an intersection) out of all n sensors. All the combinations of three sensors are gone through. If a combination is linearly independent, the intersection is calculated. Otherwise, since an intersection cannot be calculated if the sensors are linearly dependent, the total number of intersections is decreased by one. All the calculated intersections constitute a polygon with N – (number of linearly dependent combinations vertices). Lastly, the centroid of the produced polygon is taken as the position estimate.

This algorithm is similar to the one described in [12]. But here, there are restrictions on where the sensors and objects are located.

5.1.1.3 Weighted Centroid Trilateration

This algorithm is an extension of "Centroid Trilateration for 1 Object".

Two Dimensions:

The same procedure is used here as in "Centroid Trilateration for 1 Object" in two dimensions, with the exception that the centroid taken as position estimate is weighted. The weight of a vertex in the resulting polygon, i.e. one of the intersections, is based on three parameters

1. The relative position of the object with respect to the two sensors.
2. The distance between the two sensors.
3. The distance to the object.

The first parameter, w_1, is calculated as

$$w_1 = \frac{\min(d_1, d_2)}{\max(d_1, d_2)} \tag{5.21}$$

where d_1 is the distance between the first sensor and the object and d_2 is the distance between the second sensor and the object. The second, w_2, is calculated as

$$w_2 = \frac{|x_1 - x_2|}{width} \tag{5.22}$$

where x_1 and x_2 are the x-coordinates for sensor one and sensor two, respectively and $width$ is the maximum separation between two sensors. The third, w_3, is calculated as

$$w_3 = 1 - \frac{d}{range} \tag{5.23}$$

where

$$d = \sqrt{(x_i - x_c)^2 + (y_i - y_c)^2} \tag{5.24}$$

$$x_c = \frac{x_1 + x_2}{2} \tag{5.25}$$

$$y_c = \frac{y_1 + y_2}{2} \tag{5.26}$$

$range$ is the maximum distance between an object and a sensor for which the object can be detected by the sensor, x_i is the x-coordinate for the vertex and y_i is the y-coordinate for the vertex. The total weight, w, is then calculated as

$$w = w_1 \cdot w_2 \cdot w_3 \tag{5.27}$$

The weights for all the vertices, w_i, i=1, 2, ..., *N*, are now normalized as

$$nw_i = \frac{w_i}{\sum_{j=1}^{N} w_j} \tag{5.28}$$

where nw_i is the normalized weight for the i:th vertex. The position estimate is now the weighted centroid, calculated as

$$[x, y] = [\sum_{i=1}^{N} nw_i \cdot x_i \,, \sum_{i=1}^{N} nw_i \cdot y_i] \tag{5.29}$$

where x is the x-coordinate of the position estimation, y is the y-coordinate of the position estimation and x_i and y_i are the x- and y-coordinates of the i:th vertex, respectively.

Three Dimensions:

The method is basically the same in three dimensions as in two dimensions. In three dimensions, however, the parameters that the weight of a vertex is based on are calculated differently as well as there being a fourth parameter. The fourth parameter describes how linearly dependent three

sensors giving rise to a vertex are. The reason for having this parameter is that the more linearly dependent the sensors are, the larger estimation error. As stated in section 5.1.1.2, if the sensors are linearly dependent, there position cannot be estimated. The parameter w_1 is calculated as

$$w_1 = \frac{\min(d_1, d_2, d_3)}{\max(d_1, d_2, d_3)} \tag{5.30}$$

where d_1, d_2 and d_3 are the distances for sensors one, two and three to the object, respectively. The parameter w_2 is calculated, by help of Heron's formula and the law of cosines, as

$$w_2 = \frac{a_t}{a_m} \tag{5.31}$$

where

$$a_t = \sqrt{s(s - s_1)(s - s_2)(s - s_3)} \tag{5.32}$$

$$s = \frac{s_1 + s_2 + s_3}{3} \tag{5.33}$$

$$s_1 = \sqrt{(x_2 - x_3)^2 + (y_2 - y_3)^2} \tag{5.34}$$

$$s_2 = \sqrt{(x_1 - x_3)^2 + (y_1 - y_3)^2} \tag{5.35}$$

$$s_3 = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \tag{5.36}$$

a_m is the area where the sensors can be placed, x_1, x_2 and x_3 are the x-coordinates for sensors one, two and three, respectively and y_1, y_2 and y_3 are the y-coordinates for sensors one, two and three, respectively. The parameter w_3 is calculated as

$$w_3 = 1 - \frac{d}{range} \tag{5.37}$$

where

$$d = \sqrt{(x_i - x_c)^2 + (y_i - y_c)^2 + (z_i - z_c)^2} \tag{5.38}$$

$$x_c = \frac{x_1 + x_2 + x_3}{3} \tag{5.39}$$

$$y_c = \frac{y_1 + y_2 + y_3}{3} \tag{5.40}$$

$$z_c = 0 \tag{5.41}$$

$range$ is the maximum distance between an object and a sensor for which the object can be detected by the sensor, x_i, y_i and z_i are the x-, y- and z-coordinates, respectively, for the vertex being weighted. The last parameter, w_4, is calculated, by help of the law of cosines, as

$$w_4 = \frac{3\alpha}{\pi} \tag{5.42}$$

where

$$\alpha = \min(\alpha_1, \alpha_2, \alpha_3) \tag{5.43}$$

$$\alpha_1 = \cos^{-1}\left(\frac{s_2^2 + s_3^2 - s_1^2}{2s_2 s_3}\right) \tag{5.44}$$

$$\alpha_2 = \cos^{-1}\left(\frac{s_1^2 + s_3^2 - s_2^2}{2s_1 s_3}\right) \tag{5.45}$$

$$\alpha_3 = \cos^{-1}\left(\frac{s_1^2 + s_2^2 - s_3^2}{2s_1 s_2}\right) \tag{5.46}$$

The rest of the algorithm is exactly the same as in the two dimensional case.

These two algorithms are quite similar to those in [8] and [13]. The difference is that in this thesis there are restrictions on the sensors placement as well as the fact that the second algorithm is used in three dimensions and not only two.

5.1.2 Statistical

The statistical approaches in this Thesis are based on the Maximum Likelihood method, which is described below.

Since the distance estimates, d_i, are noisy, they can be modeled as:

$$d_i = f_i + n_i, i = 1, \dots, N \tag{5.47}$$

where N is the number of distance estimates (i.e. sensors), f_i is the exact distance from the i:th sensor to the object and n_i is the noise for the i:th distance estimation. Now, the Maximum Likelihood (ML) method can be applied. The ML method takes as estimate the position of the object, represented by θ, that makes our distance estimates as probable as possible [10]:

$$\hat{\theta} = \arg\max(L(\theta) = p(d|\theta)) \tag{5.48}$$

where $\hat{\theta}$ is the estimate of the position θ, $L(\theta)$ is the likelihood function and $p(d|\theta)$ is the probability density function of $d = d_i, i = 1, \dots, N$ conditioned on θ. f_i is a deterministic function and hence the likelihood function can be expressed as the probability function of the noise instead. Since the noise is zero mean Gaussian noise (stated in section 1.1) and it can be assumed to be the same for all distance estimates, the likelihood function can be expressed as:

$$p(d|\theta) = \frac{1}{(2\pi)^{N/2}\sigma^N} e^{\left(-\sum_{i=1}^{N}\frac{(d_i - f_i)^2}{2\sigma^2}\right)} \tag{5.49}$$

where σ is the standard deviation for the noise. The position estimate, $\hat{\theta}$, can now be obtained as:

$$\hat{\theta} = \arg\min(g = \sum_{i=1}^{N}(d_i - f_i)^2) \tag{5.50}$$

This estimator is known as the non-linear least squares estimator. [4][9]

5.1.2.1 Gradient Descent

One way to solve the equation above is to use a gradient descent algorithm. The gradient descent is used to find a local minimum for a certain function, in this case g. The method starts at an initial guess of the local minimum and then steps, with a certain step size γ, in the direction of the negative gradient of g at the initial guess to a new point. At this new point the process is repeated. The algorithm runs until a certain condition is fulfilled; for example, until the step size is sufficiently small.

Two Dimensions:

f_i can be expressed as:

$$f_i(x, y) = \sqrt{(x - x_i)^2 + (y - y_i)^2} \tag{5.51}$$

where x is the x-coordinate of the object, y is the y-coordinate of the object, x_i is the x-coordinate of the i:th sensor and y_i is the y-coordinate of the i:th sensor.

The algorithm implemented for the two dimensional case uses one of the intersect points as an initial guess for the gradient descent. This intersect point can be found using the geometrical trilateration method in two dimensions described above. Then the gradient of g, ∇g, is calculated as:

$$\nabla g = \begin{bmatrix} \frac{dg}{dx} = -2\sum_{i=1}^{N}\left(d_i - f_i(x,y)\right) \cdot \frac{df_i(x,y)}{dx} = 2\sum_{i=1}^{N} x - x_i - \frac{d_i \cdot (x - x_i)}{f_i(x,y)} \\ \frac{dg}{dy} = -2\sum_{i=1}^{N}\left(d_i - f_i(x,y)\right) \cdot \frac{df_i(x,y)}{dy} = 2\sum_{i=1}^{N} y - y_i - \frac{d_i \cdot (y - y_i)}{f_i(x,y)} \end{bmatrix} \tag{5.52}$$

The new point, p_{new}, is the old point, p_{old}, minus $\gamma \cdot \nabla g(p_{old})$. The algorithm, as explained above, runs until the step size is smaller than a certain precision. The x- and y-value obtained are the position estimates.

Three Dimensions:

f_i can be expressed as:

$$f_i(x,y,z) = \sqrt{(x-x_i)^2 + (y-y_i)^2 + (z-z_i)^2} \tag{5.53}$$

where z is the z-coordinate of the object and z_i is the z-coordinate of the i:th sensor.

The algorithm implemented is the same as for the two dimensional case with the exception being that the gradient of g, ∇g, is:

$$\nabla g = \begin{bmatrix} \frac{dg}{dx} = 2\sum_{i=1}^{N} x - x_i - \frac{d_i \cdot (x-x_i)}{f_i(x,y,z)} \\ \frac{dg}{dy} = 2\sum_{i=1}^{N} y - y_i - \frac{d_i \cdot (y-y_i)}{f_i(x,y,z)} \\ \frac{dg}{dz} = 2\sum_{i=1}^{N} z - z_i - \frac{d_i \cdot (z-z_i)}{f_i(x,y,z)} \end{bmatrix} \tag{5.54}$$

5.1.2.2 Brute Force

If, however, the area or volume being searched is small enough, a brute force method can be used to solve the non-linear least squares estimator. The brute force approach defines an area or volume with a certain resolution (i.e. steps in the two- or three-dimensional matrix). This defined area or volume is then put in as the x- y- and z-values in equation 3.50. After this, a simple search for the minimum value is conducted in order to find the position of the object. This method is in general the most stable and has the highest accuracy if the resolution is high enough. However, the major drawback of this method is the possibly very high computational complexity, for example if the area or volume is large or the resolution is very high.

5.1.2.3 Linear Least Squares Estimation:

Instead of trying to solve the non-linear system (5.50) in section 5.1.2, there are methods to linearize the system. One of the most common methods is to create linear lines of positions instead of circular lines of positions [12], see figure 19 below.

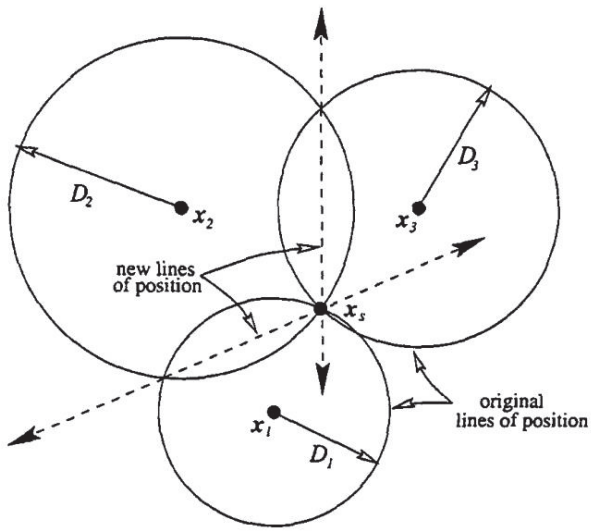


Figure 19. Linear lines of position as well as circular (original) lines of position for trilateration [12]

Actually this is not technically a linearization but it has the same effect. To produce these linear lines of positions the first *N*-1 distance formulas for the different sensors are subtracted by the *N*:th, where *N* is the number of sensors. This is similar to the geometric approach above, but here the number of sensors has to be more than the dimension, i.e. at least three sensors in two dimensions and at least four sensors in three dimensions. However, there is no upper limit for the amount of sensors. An example now follows for $N = 4$ in three dimensions. The distance formulas are:

$$d_1^2 = (x - x_1)^2 + (y - y_1)^2 \tag{5.55}$$

$$d_2^2 = (x - x_2)^2 + (y - y_2)^2 \tag{5.56}$$

$$d_3^2 = (x - x_3)^2 + (y - y_3)^2 \tag{5.57}$$

$$d_4^2 = (x - x_4)^2 + (y - y_4)^2 \tag{5.58}$$

Subtracting the last equation produces the linear system [7]:

$$2(x_4 - x_1) + 2(y_4 - y_1) = d_1^2 - d_4^2 - x_1^2 - y_1^2 + x_4^2 + y_4^2 \tag{5.59}$$

$$2(x_4 - x_2) + 2(y_4 - y_2) = d_2^2 - d_4^2 - x_2^2 - y_2^2 + x_4^2 + y_4^2 \tag{5.60}$$

$$2(x_4 - x_3) + 2(y_4 - y_3) = d_3^2 - d_4^2 - x_3^2 - y_3^2 + x_4^2 + y_4^2 \tag{5.61}$$

which can be written in matrix form as

$$Ax = b \tag{5.62}$$

This is a general form for the linear lines of positions regardless of the number of sensors (at least three) or the dimension used. In the general case A is a $(N-1) \times D$ matrix (D is the dimension), x is the position of the object (for example $x = [x, y, z]^T$ in three dimensions) and b is a vector of same size as x. The system can be solved using a Least Squares method [14]:

$$\hat{x} = (A^T A)^{-1} \cdot A^T b \tag{5.63}$$

However, the criterion for this solution is for the columns in A to be linearly independent which is not the case in this thesis. As stated in the Problem Definition (section 1.1), the sensors are distributed along the x-axis in two dimensions and the sensors are located in the xy-plane in three dimensions. Therefore, this method has not been implemented.

5.1.3 Error Analysis

In order to evaluate the position error, the distance estimation errors' impact on the position error will be considered in this section.

This is done by taking expressions of the positions, (x, y, z), as functions of the distance estimations, d_i, and the sensor positions, (x_i, y_i, z_i), linearize these with respect to the distance estimations and then calculate the position errors, $(\Delta x, \Delta y, \Delta z)$.

If the function for the x-position is denoted $f(d_1, d_2, \dots, d_N)$, then, by linearization, the absolute error of the x-position, Δx, can be expressed by

$$\Delta x = \sqrt{\sum_{i=1}^{N} \left(\frac{\partial f(d_1, d_2, \dots, d_N)}{\partial d_i} \cdot \Delta d_i\right)^2} \tag{5.64}$$

where N is the number of sensors (i.e. number of distance estimations), d_i is the distance estimation for the i:th sensor and Δd_i is the distance estimation error for the i:th sensor. The same linearization is done to retrieve the y- and z-position errors:

$$\Delta y = \sqrt{\sum_{i=1}^{N} \left(\frac{\partial g(d_1, d_2, \dots, d_N)}{\partial d_i} \cdot \Delta d_i\right)^2} \tag{5.65}$$

$$\Delta z = \sqrt{\sum_{i=1}^{N} \left(\frac{\partial h(d_1, d_2, \dots, d_N)}{\partial d_i} \cdot \Delta d_i\right)^2} \tag{5.66}$$

where $g(d_1, d_2, \dots, d_N)$ is the function for the y-position and $h(d_1, d_2, \dots, d_N)$ is the function for the z-position. [7]

Two Dimensions:

In two dimensions, $f(d_1, d_2, \dots, d_N)$ and $g(d_1, d_2, \dots, d_N)$ is already derived in section 5.1.1, namely equations (5.5) and (5.6), respectively. From these equations, the derivatives can be calculated as

$$\frac{\partial f(d_1, d_2)}{\partial d_1} = \frac{2d_1}{2(x_2 - x_1)} \tag{5.67}$$

$$\frac{\partial f(d_1, d_2)}{\partial d_2} = \frac{2d_2}{2(x_2 - x_1)} \tag{5.68}$$

and

$$\frac{\partial g(d_1, d_2)}{\partial d_1} = \frac{d_1 - (f(d_1, d_2) - x_1) \cdot \frac{\partial f(d_1, d_2)}{\partial d_1}}{g(d_1, d_2)} \tag{5.69}$$

$$\frac{\partial g(d_1, d_2)}{\partial d_2} = -\frac{(f(d_1, d_2) - x_1) \cdot \frac{\partial f(d_1, d_2)}{\partial d_2}}{g(d_1, d_2)} \tag{5.70}$$

By putting equations (5.67) and (5.68) into (5.64) as well as equations (5.69) and (5.70) into (5.65), the position errors can easily be calculated.

Three Dimensions:

The approach is the same in three dimensions; $f(d_1, d_2, \dots, d_N)$, $g(d_1, d_2, \dots, d_N)$ and $h(d_1, d_2, \dots, d_N)$ is derived from equations (5.14) and (5.15) in section 5.1.1:

$$x = f(d_1, d_2, \dots, d_N)$$

$$= \frac{(d_2^2 - d_3^2 - x_2^2 - y_2^2 + x_3^2 + y_3^2).(y_3 - y_1) - (d_1^2 - d_3^2 - x_1^2 - y_1^2 + x_3^2 + y_3^2) \cdot (y_3 - y_2)}{2(x_3 - x_2) \cdot (y_3 - y_1) - 2(x_3 - x_1) \cdot (y_3 - y_2)} \tag{5.71}$$

$$y = g(d_1, d_2, \dots, d_N)$$

$$= \frac{(d_2^2 - d_3^2 - x_2^2 - y_2^2 + x_3^2 + y_3^2).(x_3 - x_1) - (d_1^2 - d_3^2 - x_1^2 - y_1^2 + x_3^2 + y_3^2) \cdot (x_3 - x_2)}{2(y_3 - y_2) \cdot (x_3 - x_1) - 2(y_3 - y_1) \cdot (x_3 - x_2)} \tag{5.72}$$

$$z = h(d_1, d_2, \dots, d_N) = \pm\sqrt{d_1^2 - (f(d_1, d_2, \dots, d_N) - x_1)^2 - (g(d_1, d_2, \dots, d_N) - y_1)^2} \tag{5.73}$$

The various derivatives can then be calculated as

$$\frac{\partial f(d_1, d_2, d_3)}{\partial d_1} = \frac{-2(y_3 - y_2)d_1}{2(x_3 - x_2)(y_3 - y_1) - 2(x_3 - x_1)(y_3 - y_2)} \tag{5.74}$$

$$\frac{\partial f(d_1, d_2, d_3)}{\partial d_2} = \frac{2(y_3 - y_1)d_2}{2(x_3 - x_2)(y_3 - y_1) - 2(x_3 - x_1)(y_3 - y_2)} \tag{5.75}$$

$$\frac{\partial f(d_1,d_2,d_3)}{\partial d_3} = \frac{2(y_3-y_2)d_3 - 2(y_3-y_1)d_3}{2(x_3-x_2)(y_3-y_1) - 2(x_3-x_1)(y_3-y_2)} \quad (5.76)$$

$$\frac{\partial g(d_1,d_2,d_3)}{\partial d_1} = \frac{-2(x_3-x_2)d_1}{2(y_3-y_2)(x_3-x_1) - 2(y_3-y_1)(x_3-x_2)} \quad (5.77)$$

$$\frac{\partial g(d_1,d_2,d_3)}{\partial d_2} = \frac{2(x_3-x_1)d_2}{2(y_3-y_2)(x_3-x_1) - 2(y_3-y_1)(x_3-x_2)} \quad (5.78)$$

$$\frac{\partial g(d_1,d_2,d_3)}{\partial d_3} = \frac{2(x_3-x_2)d_3 - 2(x_3-x_1)d_3}{2(y_3-y_2)(x_3-x_1) - 2(y_3-y_1)(x_3-x_2)} \quad (5.79)$$

$$\frac{\partial h(d_1,d_2,d_3)}{\partial d_1} = \frac{d_1 - (f-x_1)\cdot\frac{\partial f}{\partial d_1} - (g-y_1)\cdot\frac{\partial g}{\partial d_1}}{h} \quad (5.80)$$

$$\frac{\partial h(d_1,d_2,d_3)}{\partial d_2} = \frac{-(f-x_1)\cdot\frac{\partial f}{\partial d_2} - (g-y_1)\cdot\frac{\partial g}{\partial d_2}}{h} \quad (5.81)$$

$$\frac{\partial h(d_1,d_2,d_3)}{\partial d_3} = -\frac{-(f-x_1)\cdot\frac{\partial f}{\partial d_3} - (g-y_1)\cdot\frac{\partial g}{\partial d_3}}{h} \quad (5.82)$$

Where f is $f(d_1,d_2,d_3)$, g is $g(d_1,d_2,d_3)$ and h is $h(d_1,d_2,d_3)$. When inserting equations (5.74) – (5.82) above into equations (5.64) – (5.66) the position errors can easily be calculated.

5.2 Average Runtime for Brute Force Algorithm

As described in section 5.1.2.2, the accuracy of the Brute Force algorithm is heavily dependent on the resolution parameter. However, the quite poor time complexity of the algorithm limits this parameter. Due to this, a further evaluation of the runtime of the algorithm is needed.

5.2.1 Test Setup

Two Dimensions:

20 simulations are performed, where the resolution for the algorithm changes. The algorithm conducts its search within the Area. The sensors are placed on the x-axis at [-0.05 -0.017 0 0.017 0.05]. In a simulation, 1000 iterations are performed.

In each iteration, one object is randomly placed within the Area. The distances between the sensors and the object are generated with zero mean Gaussian noise with a standard deviation of 1 mm. The runtime for the position estimation is saved.

After the simulations the average runtimes for the different resolutions are calculated and plotted.

Three Dimensions:

20 simulations are performed, where the resolution for the algorithm changes. The algorithm conducts its search within the Volume. The sensors are placed in the xy-plane at the positions (-0.05 -0.05 0) (-0.05 0.05 0) (0 0 0) (0.05 0.05 0) (0.05 -0.05 0). In a simulation, 1000 iterations are performed.

In each iteration, one object is randomly placed within the Volume. The distances between the sensors and the object are generated with zero mean Gaussian noise with a standard deviation of 1 mm. The runtime for the position estimation is saved.

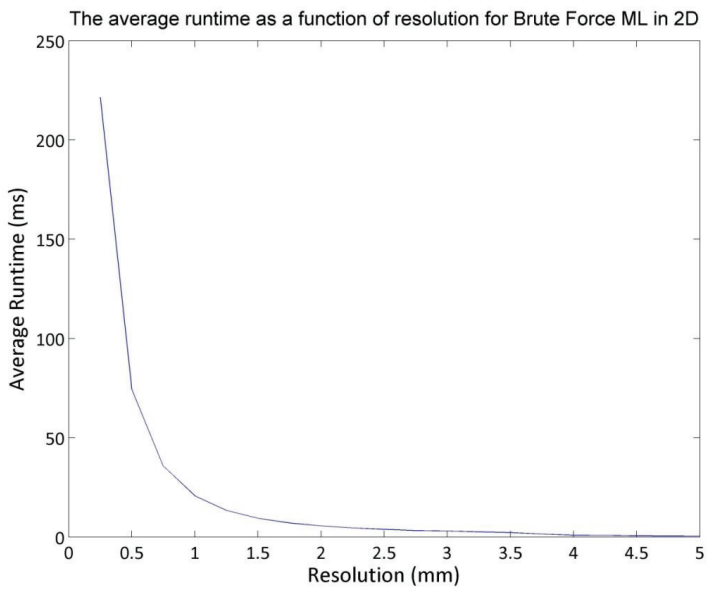
After the simulations the average runtimes for the different resolutions are calculated and plotted.

5.2.2 Results

The results are presented as plots of average runtime versus resolution (i.e. how fine steps the two- or three-dimensional matrix has in which the search is done).

Two Dimensions:

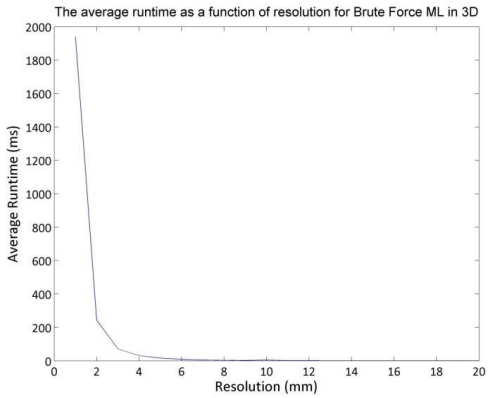
Plot 15 shows how the average runtime depends on the resolution of the area to be searched.



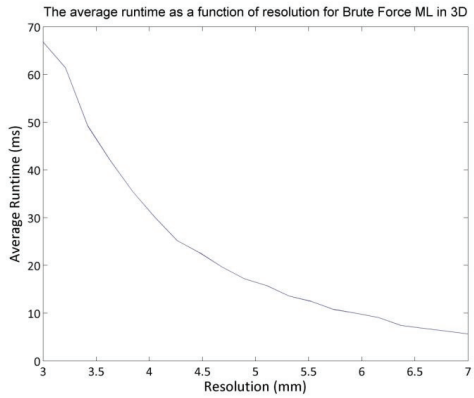
Plot 15. Average runtime as a function of resolution in two dimensions

Three Dimensions:

Both the plots below show the average runtime versus resolution of the volume to be searched. The plot on the right-hand side is a zoom-in of the plot on the left-hand side, with the values on the x-axis ranging between 3.0 and 7.0 mm.



Plot 16. Average runtime as a function of resolution in three dimensions



Plot 17. Zoom-in of plot 16

5.2.3 Discussion and Conclusions

As expected and can be seen in plot 15, 16 and 17, the Brute Force algorithm has very steep curves regarding the runtime as a function of resolution. It is expected since in two dimensions the time complexity is $O(n^2)$ due to that it is an area to be searched and three dimensions the time complexity is $O(n^3)$ due to that it is a volume to be searched.

If one wants to be able to update the position of an object relatively fast in two dimensions, the resolution cannot be much higher than 0.5 mm. With this resolution the Brute Force algorithm has an average runtime of 75 ms (see plot 15) which gives rise to a maximum update speed of around 13 Hz.

If in three dimensions (which is more likely when having a sensing application in a mobile phone) the resolution cannot be much higher than 3 mm if one wants to be able to update the position estimation at a speed of around 15 Hz.

In other words, as described in section 5.1, the Brute Force algorithm is generally the best in terms of stability and accuracy but it very important to know the limitations. For example, estimating a position with sub millimeter accuracy with this method is almost not feasible in three dimensions in a mobile application where everything is needs to be fast to enhance user experience.

5.3 Impact of Distance Error on Position Error

A point of evaluation to be conducted in this Thesis, according to section 1.1, is the impact of the distance error on the position error. This is done below.

5.3.1 Test Setup

Two Dimensions:

In the test, 10 000 simulations are conducted.

In each simulation, one object is randomly placed within the Area. Since the sensors should be placed as far away from each other as possible, see section 4.1, two sensors are placed at x=-0.05 m and x=0.05 m and the distance error is changed from 0 m to 0.01 m. The position error for each distance error is calculated (using the algorithm in section 5.1.3) and saved.

After the simulations, the average position error is plotted against the distance error.

Three Dimensions:

In the test, 10 000 simulations are conducted.

In each simulation, one object is randomly placed within Volume. Since the area of the triangle with the modules as vertices should be as large as possible, see section 4.1, three modules are placed in the xy-plane at (0 0.05 0), (-0.05 -0.05 0) and (0.05 -0.05 0). The distance error is changed from 0 m to 0.01 m and the position error for each distance error is calculated (using the algorithm in section 5.1.3) and saved.

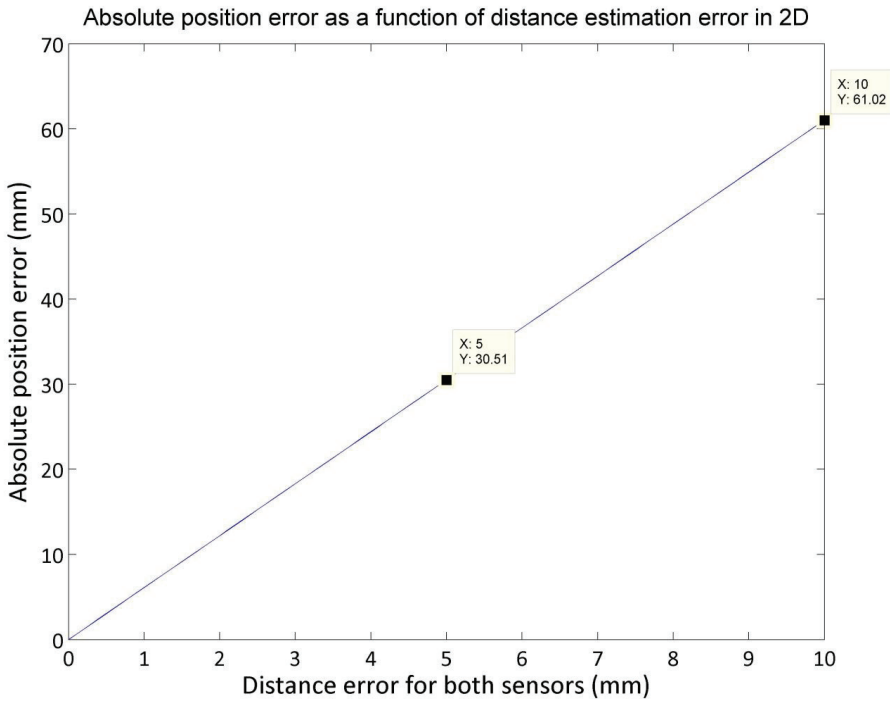
After the simulations, the average position error is plotted against the distance error.

5.3.2 Results

The results are displayed as absolute position error versus distance error plots.

Two Dimensions:

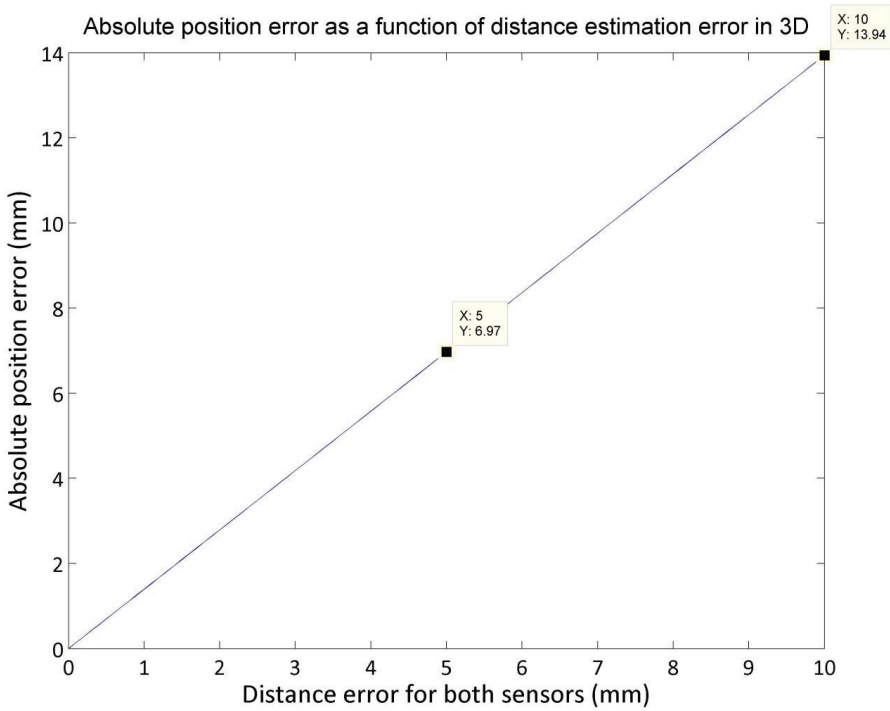
Below, in plot 18, the absolute position error as a function of distance estimation error in two dimensions is presented.



Plot 18. Position error as a function of distance error in two dimensions

Three Dimensions:

The corresponding plot in three dimensions is presented below.



Plot 19. Position error as a function of distance error in three dimensions

5.3.3 Discussion and Conclusions

Two Dimensions:

As can be seen in the two plots above, the position error is linearly dependent on the distance estimation error. This is however obvious, since the algorithm used to calculate the position error uses a linearization technique, see section 5.1.3. Hence, the shape of the plot does not convey a lot of information. However, the slope can easily be calculated and a "transition factor", *TF*, can be determined as follows:

$$TF = \frac{\Delta y}{\Delta x} = \frac{61.02 - 30.53}{10 - 5} \approx 6.1 \tag{5.83}$$

This factor states how much worse the absolute position error becomes due to a distance estimation error. Note here that this factor is highly dependent on both sensor placement and sensor density and hence is not an absolute value.

Three Dimensions:

Just as in the two dimensional case, the position error is linearly dependent on the distance estimation error. Again, this is due to the linearization technique used in the algorithm that calculates the position error. However, the transition factor, *TF,* is quite a bit lower compared to the two dimensional case:

$$TF = \frac{\Delta y}{\Delta x} = \frac{13.81 - 6.904}{10 - 5} \approx 1.4 \tag{5.84}$$

Again, this factor is highly dependent on sensor placement and sensor density and thus the actual position error in three dimensions may well be worse than in two dimensions even if the distance estimation errors are the same.

5.4 Position Estimation

This section evaluates the algorithms for the position estimation for one object. The sensors in this section are placed according to section 4.1 as far apart as possible. In a mobile phone this limit would be around 10 cm and thus the sensors are placed accordingly. The number of sensors is four in two dimensions and five in three dimensions. This is obviously a tradeoff; a high number of sensors increases the accuracy but also increase the cost.

5.4.1 Test Setup

Two Dimensions:

4 simulations are performed, where the strength of the noise is changed. The sensors are placed on the x-axis at -0.05 m, -0.025 m, 0.025 m and 0.05 m. In a simulation, 1000 iterations are performed.

In each iteration, one object is randomly placed within the Area. The distances between the sensors and the object are generated with a zero mean Gaussian noise. The position of the object is then estimated using the four implemented algorithms:

1. Geometrical Centroid
2. Geometrical Weighted Centroid
3. Statistical Maximum Likelihood using Gradient Descent
4. Statistical Maximum Likelihood using Brute Force

The runtimes as well as the position errors for the four algorithms are calculated and saved.

After a simulation, cumulative density functions for the errors are plotted and some other statistics, such as the standard deviation, are calculated.

Three Dimensions:

The setup is the same as for the two dimensional case, except that the objects is located within the Volume instead of the Area. Also, the sensors are instead placed in the xy-plane at (-0.05 -0.05 0) (-0.05 0.05 0) (0 0 0) (0.05 0.05 0) (0.05 -0.05 0).

The resolution for the Brute Force algorithm is 0.75 mm in two dimensions and 4 mm in three dimensions due to this being a good tradeoff between accuracy and time complexity, see section 5.2.

5.4.2 Results

As for the time delay estimation (section 3.3), the results are divided into three parameters; stability, accuracy and computational complexity. First, the two-dimensional results are presented and then the three-dimensional results. All results have four different standard deviations of the added noise (this is noise added to the generated distances that serves as input for the algorithms), ranging from 0.1 to 10 mm.

Two Dimensions:

Stability

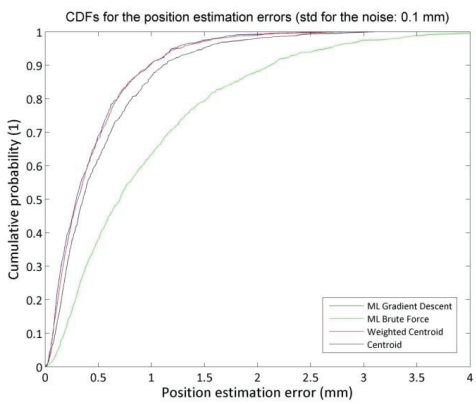
The stability is here evaluated based on how many positions that weren't found, as can be seen in the table below.

Algorithm	Std of 0.1 mm	Std of 1.0 mm	Std of 5.0 mm	Std of 10 mm
Centroid	0 %	0 %	0 %	0 %
Weighted Centroid	0 %	0 %	0 %	0 %
ML Brute Force	0 %	0 %	0 %	0 %
ML Gradient Descent	0 %	0 %	0 %	0 %

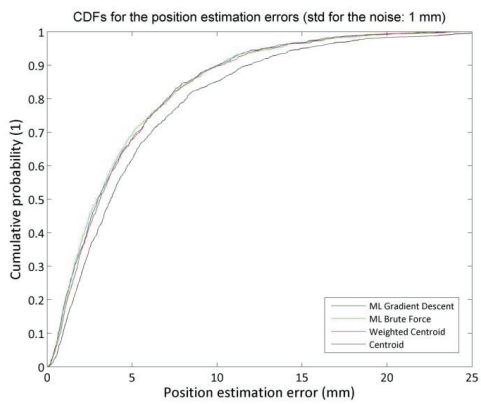
Table 7. Percentage of times a position wasn't found for the algorithms for different standard deviations of the noise (noise of the distance estimations that serves as input for the algorithms) in two dimensions

Accuracy

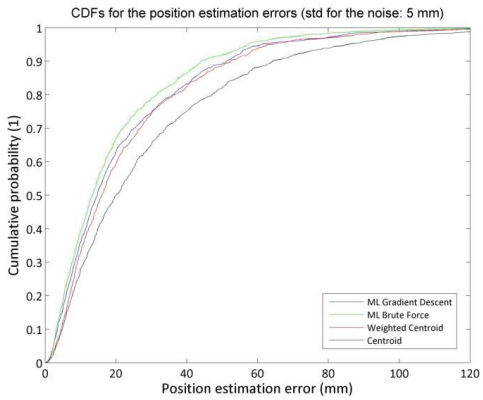
Again, as for the time delay estimation results, plots are made showing the cumulative distribution functions of the estimation errors for the algorithms in order to evaluate the accuracy. There are a total of four plots, one for each standard deviation, and they are presented below.



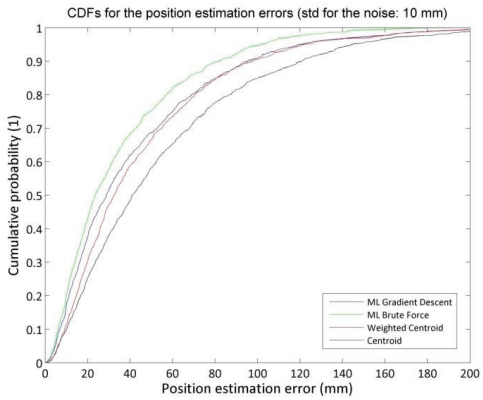
Plot 20. Cumulative distribution functions for the algorithms with noise std of 0.1 mm in two dimensions



Plot 21. Cumulative distribution functions for the algorithms with noise std of 1.0 mm in two dimensions



Plot 22. Cumulative distribution functions for the algorithms with noise std of 5.0 mm in two dimensions



Plot 23. Cumulative distribution functions for the algorithms with noise std of 10 mm in two dimensions

Computational Complexity

Lastly, to evaluate the computational complexity, the average runtime for the four algorithms are presented in table 8 below.

Algorithm	Std of 0.1 mm	Std of 1.0 mm	Std of 5.0 mm	Std of 10 mm
Centroid	0.593 ms	0.601 ms	0.629 ms	0.630 ms
Weighted Centroid	0.885 ms	0.932 ms	0.934 ms	0.924 ms
ML Brute Force	15.4 ms	15.1 ms	15.9 ms	15.5 ms
ML Gradient Descent	10.1 ms	13.9 ms	17.9 ms	19.2 ms

Table 8. Average runtimes for the algorithms for different standard deviations of the noise (noise of the distance estimations that serves as input for the algorithms) in two dimensions

Three Dimensions:

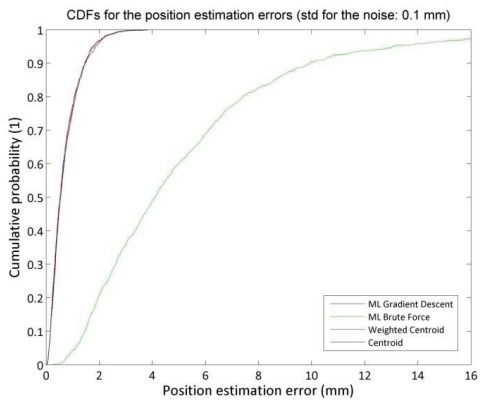
The results in three dimensions are presented exactly the same as in two dimensions, as can be seen below.

Stability

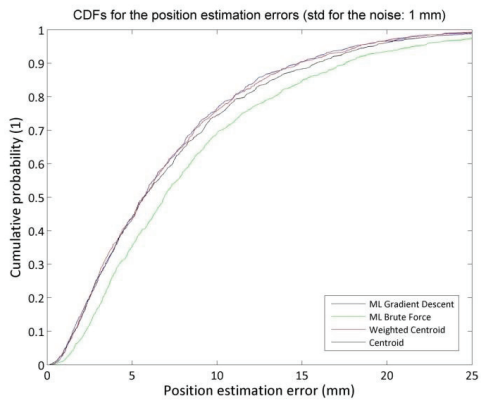
Algorithm	Std of 0.1 mm	Std of 1.0 mm	Std of 5.0 mm	Std of 10 mm
Centroid	0 %	0.1 %	0.4 %	0.8 %
Weighted Centroid	0 %	0.1 %	0.4 %	0.8 %
ML Brute Force	0 %	0 %	0 %	0 %
ML Gradient Descent	0 %	0.1 %	0.4 %	0.8 %

Table 9. Percentage of times a position wasn't found for the algorithms for different standard deviations of the noise (noise of the distance estimations that serves as input for the algorithms) in three dimensions

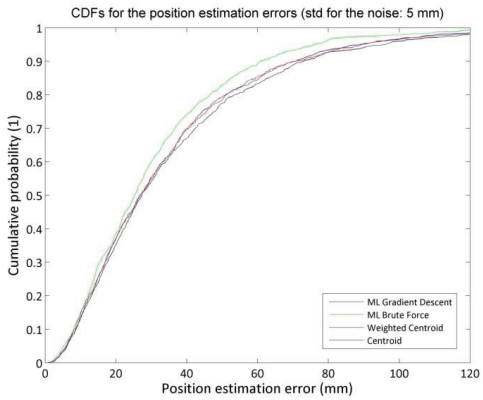
Accuracy



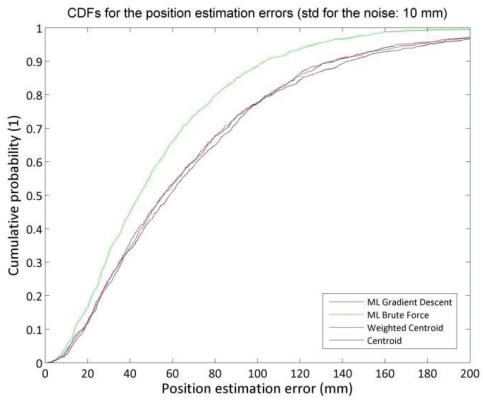
Plot 24. Cumulative distribution functions for the algorithms with noise std of 0.1 mm in three dimensions



Plot 25. Cumulative distribution functions for the algorithms with noise std of 1.0 mm in three dimensions



Plot 26. Cumulative distribution functions for the algorithms with noise std of 5.0 mm in three dimensions



Plot 27. Cumulative distribution functions for the algorithms with noise std of 10 mm in three dimensions

Computational Complexity

Algorithm	Std of 0.1 mm	Std of 1.0 mm	Std of 5.0 mm	Std of 10 mm
Centroid	0.169 ms	0.176 ms	0.192 ms	0.227 ms
Weighted Centroid	0.446 ms	0.459 ms	0.492 ms	0.565 ms
ML Brute Force	30.7 ms	31.8 ms	33.1 ms	38.1 ms
ML Gradient Descent	8.79 ms	17.1 ms	22.3 ms	30.5 ms

Table 10. Average runtimes for the algorithms for different standard deviations of the noise (noise of the distance estimations that serves as input for the algorithms) in three dimensions

5.4.3 Discussion and Conclusions

Two Dimensions:

Stability

The stability, based on how many failed position estimations were made, is good for all four algorithms as can be seen in table 7. To note here though is that the distance estimations (i.e. the input of these algorithms) have been generated and not actually estimated. Therefore, the input is perfect in the sense that there is only noise added to the distances and no other strange errors (which may be the case when actually estimating the distances).

Accuracy

From plots 20 to 23, one can see the transition of the Maximum Likelihood (ML) Brute Force algorithm from being the worst to being the best in terms of accuracy when increasing the noise. The

reason for this is the resolution parameter of the algorithm. Since the resolution is 0.75 mm, when having a noise standard deviation as small as 0.1 mm, the other algorithms have higher accuracies since they are not limited by this resolution. But if the noise is larger (have a standard deviation larger than the resolution parameter) the ML Brute Force Algorithm outperforms the other algorithms in terms of accuracy.

Another thing to note in these plots is the fact that the Centroid algorithm is consistently the worst (of course except from the ML Brute Force algorithm when having low noise input). This is expected since the Weighted Centroid is an extension or enhancement of the Centroid algorithm. And the Maximum Likelihood (ML) Gradient Descent, despite being a statistical algorithm, is also basically an extension of the Centroid algorithm since it takes intersections as start points for the descent.

The ML Gradient Descent and Weighted Centroid methods have basically the same accuracy. The ML Gradient Descent is marginally better when the noise is larger (standard deviations of 5 mm and 10 mm).

Computational Complexity

There is a very distinct difference between the two geometrical algorithms, Centroid and Weighted Centroid, and the two statistical algorithms, ML Brute Force and ML Gradient Descent when it comes to average runtimes, which can be seen in table 8. The reason that the geometrical algorithms are significantly faster is that they basically only calculates intersections and means whereas the statistical algorithms performs a search of some kind.

There is no difference in average runtime for Centroid or Weighted Centroid when increasing the noise. This is due to the fact that whatever the noise, there is still the same amount of intersections to be calculated (of course assuming the same number of sensors and objects to be found).

The runtime of the ML Brute Force algorithm is not affected by the noise either. The reason for this is that the same matrix is to be searched through regardless of the strength of the noise (assuming the same area and resolution). However, the average runtime of ML Gradient Descent is increasing with increased noise. This is because in general the more noise, the worse the initial guess (intersection) is which means that it takes a longer time for the algorithm to reach the stop condition.

Three Dimensions:

Stability:

The algorithms are a bit more susceptible to failures when estimating in three dimensions, as can be seen in table 9. The exclusion from this is the ML Brute Force algorithm. The reason for this is that the algorithm actually never fails in the sense that it always finds the best position estimation from the distance estimations. Of course if the distance estimations are very noisy, the position estimation may have large errors, but it is always found.

The reason that the geometrical algorithms fail sometimes in three dimensions but not in two is the fact that more distance estimations are needed for an intersection (three instead of two). The same reason applies to the ML Gradient Descent due to the initial guesses being intersections.

Accuracy

Just as in the two-dimensional case the ML Brute Force algorithm is the best when it comes to accuracy if the standard deviation of the noise is larger than the resolution parameter of the algorithm, which is clearly seen in plots 24 to 27.

The Centroid algorithm is still overall the worst (for noisier signals) whereas the ML Gradient Descent and Weighted Centroid are still more accurate, but not as significantly as in the two-dimensional case.

Computational Complexity

Again, just as in the two-dimensional case, there is a huge difference in the average runtimes between the geometrical and the statistical algorithms, as seen in table 10. The runtime stays basically the same for Centroid, Weighted Centroid and ML Brute Force regardless of noise for the same reasons as in two dimensions. And finally the runtime for ML Gradient Descent is increasing with increased noise.

Conclusion

In terms of stability the algorithms are quite similar, with ML Brute Force being a bit more stable compared to the others. In terms of accuracy, the ML Brute Force is overall the best but the algorithm is quite inflexible due to the resolution parameter, especially in three dimensions. Regarding the computational complexity, the geometrical algorithms are by far the better choice.

When taking all this into account the algorithm of choice for implementation in the Android application is the Weighted Centroid, mainly due to the very fast runtime and reasonable accuracy.

6 Multiobject Localization

So far, only one object has been considered when estimating the position. However, it is often desirable to be able to locate more than one object. This section is dedicated to present and evaluate one such algorithm.

6.1 Theories and Algorithms

Below is the explanation of the multiobject localization algorithm in both two and three dimensions.

Two Dimensions:

The algorithm used to localize multiple objects is essentially an extension of the Weighted Centroid algorithm (see section 5.1.1.3).

The first step is to find all the intersections for all objects and all combinations of sensors. The intersections are calculated using the algorithm in section 5.1.1.1 in two dimensions.

The second step is to find clusters of intersections which may constitute an object's position. All the intersections are gone through and for each intersection, the sensors and distances giving rise to the intersection are extracted. This is done in order to determine a suitable tolerance around the intersection using the algorithm in section 5.1.3. This tolerance is then used when going through the rest of the intersections to evaluate if these might make up a cluster. If enough intersections are found, these intersections are regarded as a true cluster (i.e. an object resides within the cluster). The minimum number of intersections that is regarded as enough to be a true cluster is (N / 2), where N is the number of sensors and 2 is the dimension, as explained in section 5.1.1.2. See figure 20 for a more easy understanding of the process.

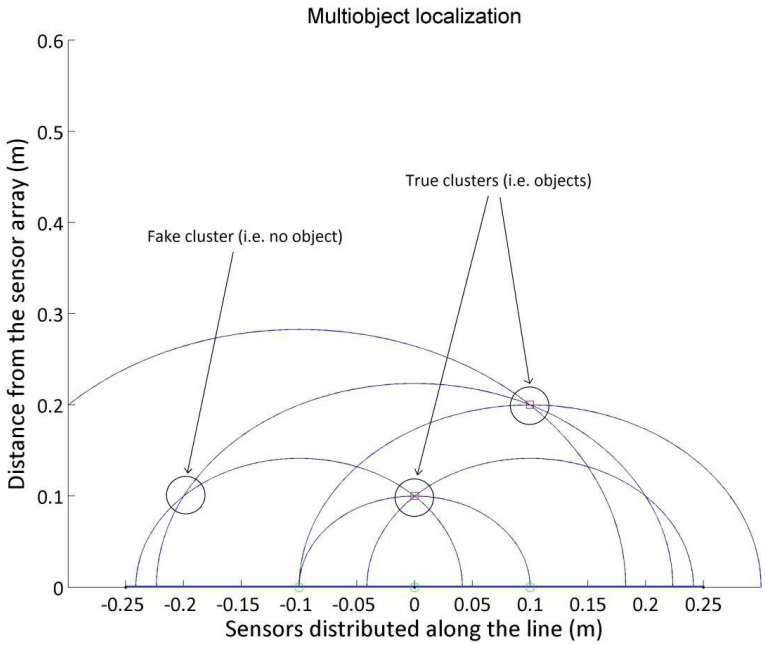


Figure 20. Multiobject localization (trilateration) with three sensors (green circles) and two objects (red squares) in two dimensions

The third step is estimating the positions from the found clusters and this is done by the Weighted Centroid algorithm.

Three Dimensions:

The algorithm in three dimensions is conducted in the same way as in two dimensions, but of course using the corresponding algorithms in sections 5.1.1.1 and 5.1.1.3 for three dimensions.

6.2 Test Setup

Again, the sensor placement and density considers both the results from section 4 as well as the cost of the sensors.

Two Dimensions:

4 simulations are done where the number of objects is changed. Three modules are placed on the x-axis at -0.05m, 0m and 0.05m. In a simulation, 10 000 iterations are executed.

In each iteration, the objects are placed randomly in the Area. Distances between the sensors and the objects are then generated containing a zero mean Gaussian noise with a standard deviation of 1 mm. The positions are estimated using the Multi Object Weighted Centroid algorithm. Lastly, relevant data, such as the estimation error and runtime, is extracted.

After the simulation, the data extracted is presented.

Three Dimensions:

The setup is the same as for the two dimensional case, the only differences is that the objects are restricted to the Volume. The sensors are located in the xy-plane at (-0.05 -0.05 0) (-0.05 0.05 0) (0.05 0.05 0) (0.05 -0.05 0).

6.3 Results

The results for multiobject localization are in the form of percentage successful estimations. The definition of a completely successful estimation is that all objects are found, the amount of objects found is correct and the estimation errors are no larger than 3 cm. The definition of a fairly successful estimation is that all objects are found, but there are no limitations on the number of objects found (i.e. the number of objects found can be larger than the actual amount of objects) or the estimation errors.

Two Dimensions:

Below is a bar chart displaying the percentage of successful estimations for different number of objects in two dimensions.

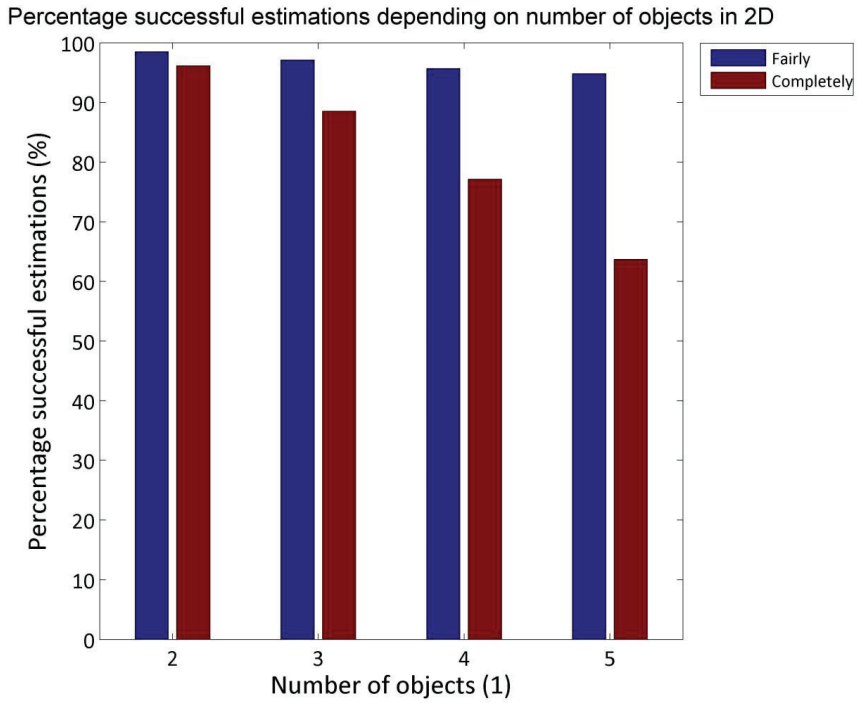
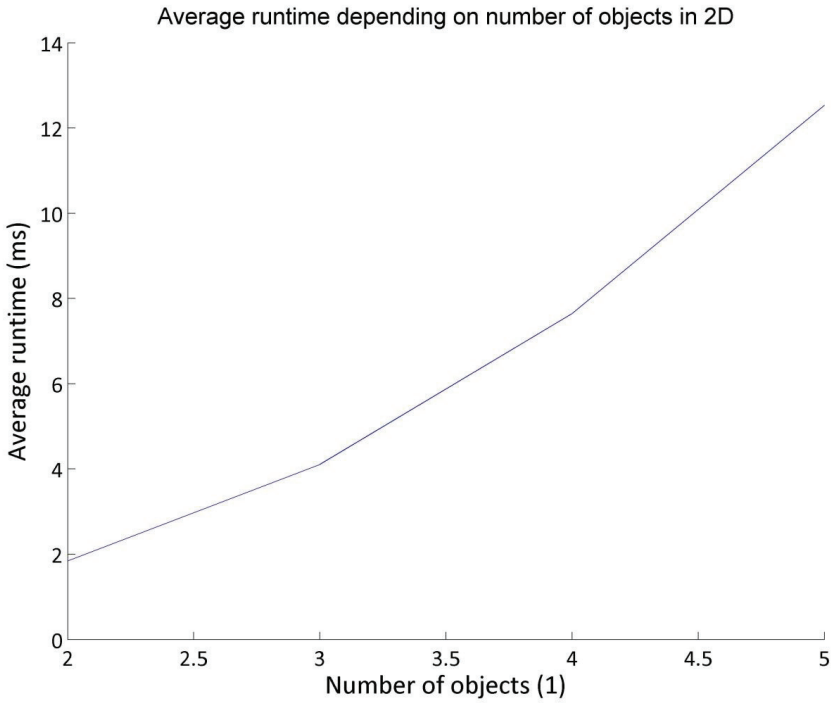


Figure 21. Bar chart displaying the percentage of successful estimations depending on number of objects to be found in two dimensions

Another interesting thing to examine is the effect on average runtime that the number of object to be found has. Below is a simple plot that displays the average runtime versus the number of objects.



Plot 28. Average runtime as a function of number of objects to be found in two dimensions

Three Dimensions:

Below is a similar chart as in figure 21 above, but in three dimensions.

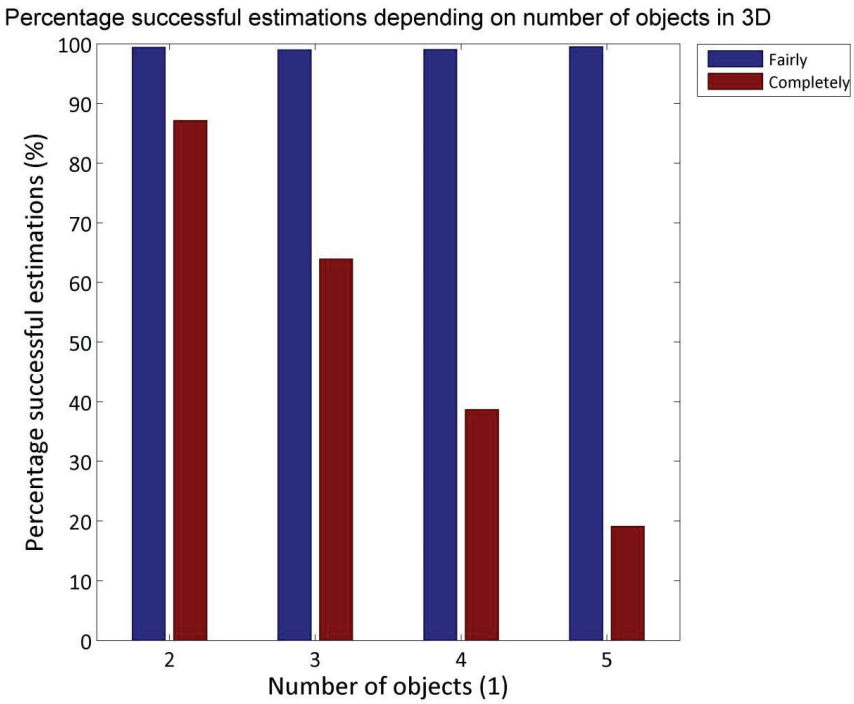
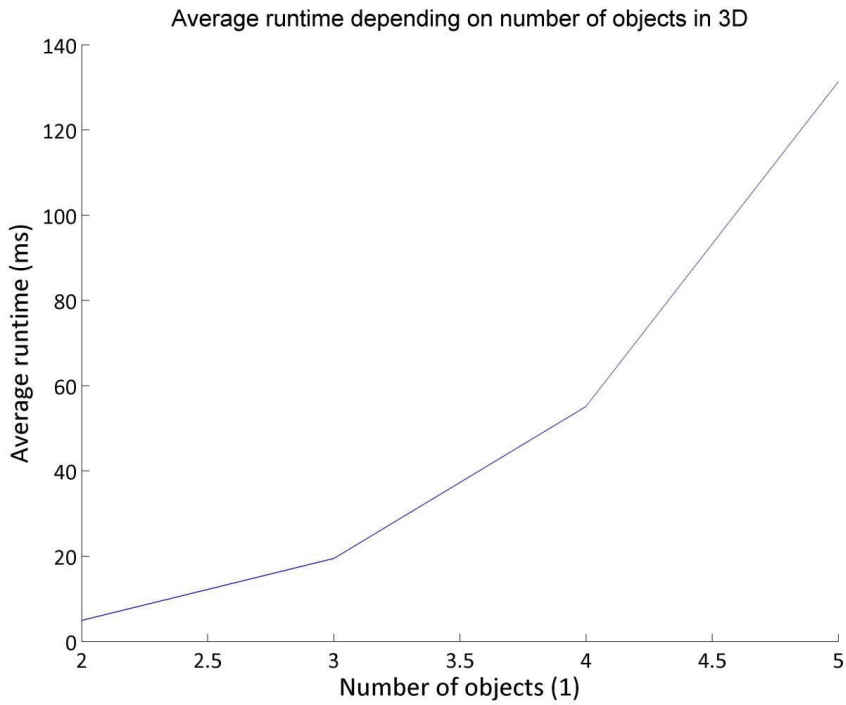


Figure 22. Bar chart displaying the percentage of successful estimations depending on number of objects to be found in three dimensions

Also, a plot of average runtime versus number of objects is shown below.



Plot 29. Average runtime as a function of number of objects to be found in three dimensions

6.4 Discussion and Conclusions

As one might suspect, the more objects to be located, the more errors occur, see figures 21 and 22. The reason for this is that when more objects are to be located, there is a larger risk that these objects reside in close proximity to each other. This fact makes the clusters harder to determine; there is a high risk that some of the intersections making up the cluster of one object is counted when making up a cluster of another object. This leads to larger errors, especially in three dimensions, and in the worst case to an object's position not being estimated.

As can be seen in plots 28 and 29, the average runtime of the localization increases as the number of objects to be found increases. This is due to the fact that more intersections need to be calculated when more objects are to be localized. If the objects are located far apart the number of intersections is increased linearly with the number of objects. But if the objects are located close to one another, additional intersections arise, see figure 20. This is the reason for the graphs' appearance in plots 28 and 29. The graph in three dimensions is steeper because more sensors are needed in three dimensions to determine the position of the objects (and hence a larger number of additional intersections need to be calculated).

What can be drawn from the results in this section is that the algorithm is very stable (i.e. very high percentage of fairly successful estimations) regardless of number of objects. This stability may of

course eventually fall off if the number of objects to be found becomes very large. The accuracy is very good for few objects but falls off when the number of objects increases, especially in three dimensions.

7 Two-Step Positioning

Up to this point, the time delay estimation and the position estimation have been evaluated separately. Of course, it is interesting to evaluate these two together, as both constitute complete object position estimation from sensor inputs. This is what is done in this section.

7.1 One Object

The evaluation of a complete two-step positioning estimation for one object is presented below.

7.1.1 Test Setup

The Covariance algorithm is used for the distance (or time delay) estimation mainly due to the stability but also the precision, as can be seen in section 3.3. All the positioning algorithms are used in order to evaluate them using a better input (i.e. an input based on TDE and not just generated). The resolution for the Brute Force algorithm is quite high; 0.5 mm. This is due to the low standard deviation of the distance estimation.

Two Dimensions:

The setup is similar to the setup for position estimation for 1 object, see section 5.4.1. The difference is that only one simulation (with 1000 iterations) is done and in each iteration, a simulated, correlated signal is generated and the distance is estimated using the covariance algorithm. From the distance estimate, the position of the object is then estimated using the four implemented algorithms.

Three Dimensions:

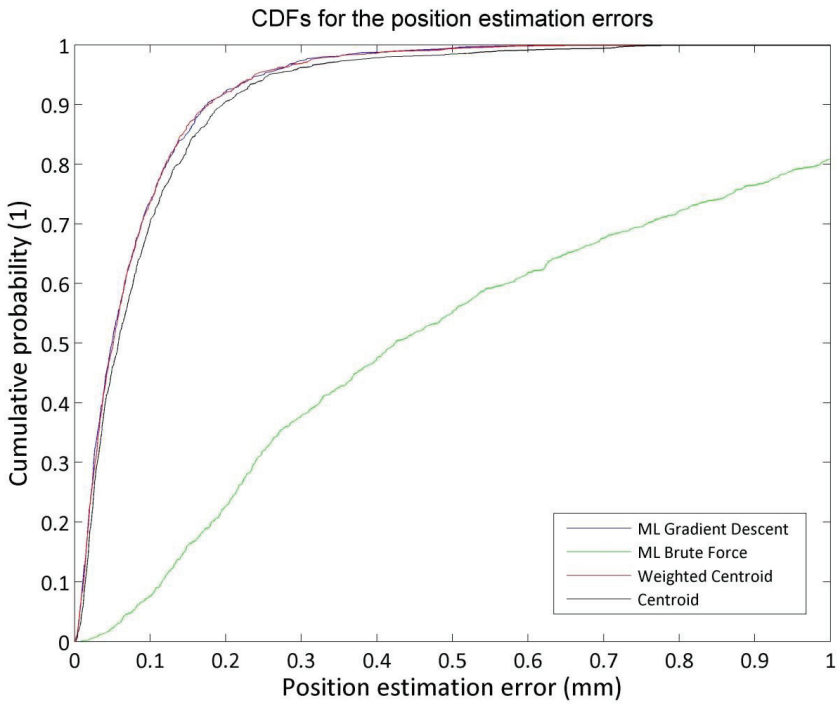
The setup is almost the same as for the two dimensional case. The difference is the removal of the Brute Force algorithm. The reason for this is that the standard deviation for the distance estimation is small (0.05mm) and hence a very high resolution is needed for the brute force algorithm. This fact makes the algorithm not feasible due to both time consumption and even memory issues. The sensors are placed in the xy-plane at (-0.05 -0.05 0) (-0.05 0.05 0) (0 0 0) (0.05 0.05 0) (0.05 -0.05 0).

7.1.2 Results

The results are presented by plotting the cumulative distribution functions of the errors for the localization algorithms. One major difference in these results compared to the time delay estimation (section 3.3) and position estimation (section 5.4) is that the definition of an object not found is stricter. Now, an object is deemed not found if the error of the time delay estimation is larger than 5 mm. The reason for this is that the simulation should be more closely related to the reality, where oftentimes a very bad estimation is worth about as much as no estimation at all.

Two Dimensions

The percentage of objects not found was 0.4 %. In plot 30, the cumulative distribution functions of the errors for the four localization algorithms are shown.



Plot 30. Cumulative distribution functions of the errors for the algorithms in two dimensions

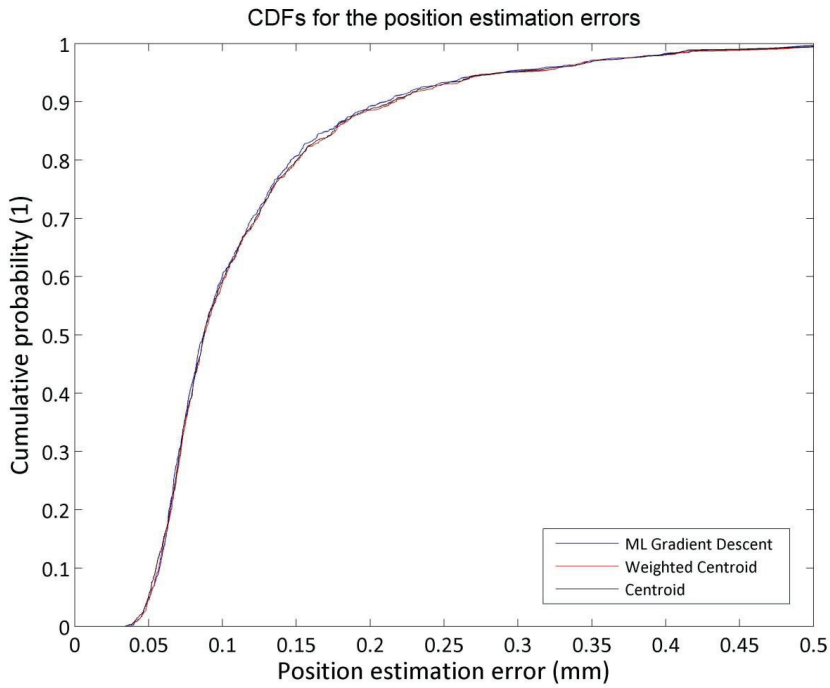
To be able to evaluate the computation complexity for the different algorithms, the table below presents the average runtimes. Note here that Covariance is the TDE algorithm whereas the rest are positioning algorithms.

Algorithm	Average runtime
Covariance	95.3 ms
Centroid	0.845 ms
Weighted Centroid	1.25 ms
ML Brute Force	44.8 ms
ML Gradient Descent	6.57 ms

Table 11. Average runtimes for the five algorithms in two dimensions

Three Dimensions:

The percentage of objects not found was now 1.0 %. As in the two-dimensional case, a cumulative distribution plot of the errors is shown below. Note that, as mentioned in the test setup (section 7.1.1), there is no plot for the ML Brute Force algorithm due to the very high computational complexity.



Plot 31. Cumulative distribution functions of the errors for the algorithms in three dimensions

Table 12 presents the average runtimes.

Algorithm	Average runtimes
Covariance	131 ms
Centroid	0.223 ms
Weighted Centroid	0.680 ms
ML Gradient Descent	5.27 ms

Table 12. Average runtimes for the four algorithms in three dimensions

7.1.3 Discussion and Conclusions

Two Dimensions:

The complete two-step positioning scheme for one object in two dimensions is found to be very stable with only 0.4 % of the objects not being localized.

Since the distance estimation errors are small with the analog signals and the high accuracy of the Covariance algorithm, the resolution parameter for the ML Brute Force algorithm is set to high (0.5 mm) but it is still not enough to be even close to perform on the same level as the other algorithms, which is evident in plot 30.

When it comes to the computational complexity, the distance estimation (i.e. the Covariance algorithm) requires the heaviest computation (see table 11). This is due to the fact that the analog signals have a very large number of data points (see section 3.4 for information). So by having input signals with significantly fewer data points (as is the case for digital input signals) the average runtime is decreased a lot. However, the localization algorithms are not dependent on the analog or digital input signals but the number of sensors used and the number of objects to be located.

The reason that the average runtime for the Covariance algorithm is around a factor four longer than in section 3.3.1 is the fact that in these tests there are four sensors whereas in the tests in section 3.3, there is only one. This linear behavior of the increase in average runtime depending on sensors was observed in section 4.2.

Three Dimensions:

In three dimensions the two-step positioning scheme is still very stable with only 1 % of the objects not being localized.

Here, again due to the small errors in the distance estimations, the ML Brute Force algorithm is not even tested, see section 7.1.1. The other algorithms have basically the same accuracy as can be seen in plot 31.

As in two dimensions it is the Covariance algorithm that has the highest computational complexity.

7.2 Multiobject

The evaluation of a complete two-step positioning estimation for more than one object is presented below.

7.2.1 Test Setup

Two Dimensions:

The setup is similar to the setup for position estimation for more than 1 object, see section 6.2.

There are three differences:

1. The number of objects is limited to 2.
2. In each simulation, a simulated, correlated signal is generated and the distance is estimated using the covariance algorithm. The position estimate is then estimated using the distance estimate.
3. The objects placed are at least 3 cm apart. The reason for this is that a correlated signal pulse is of this length and the TDE algorithm cannot handle overlapping signals.

Three Dimensions:

The setup is the same as for the two dimensional case. The sensors are located in the xy-plane at (-0.05 -0.05 0) (-0.05 0.05 0) (0.05 0.05 0) (0.05 -0.05 0).

7.2.2 Results

As for the multiobject localization (section 6.3), the results are presented as percentages successful estimations. Now, the definition of a completely successful estimation is that all distance estimation errors are no larger than 1 mm, all objects are found, the amount of objects found is correct and the position estimation errors are no larger than 3 mm. The definition of a fairly successful estimation is that all distance estimation errors are no larger than 1 cm, all objects are found, but there are no limitations on the number of objects found or the estimation errors.

Two Dimensions:

The result is presented in table 13 which shows both the successful estimations as well as the total average runtime (i.e. the average runtime for both the time delay estimation as well as the object localization).

Fairly successful estimation	Completely successful estimation	Total average runtime
93.8 %	92.9 %	53.3 ms

Table 13. Results for the complete two-step positioning procedure with two objects in two dimensions

Three Dimensions:

In table 14 below is the corresponding result in three dimensions.

Fairly successful estimations	Completely successful estimations	Total average runtime
90.4 %	78.2 %	72.5 ms

Table 14. Results for the complete two-step positioning procedure with two objects in three dimensions

7.2.3 Discussion and Conclusions

Finally the complete multiobject two-step positioning procedure is to be evaluated. The stability, both in two as well as three dimensions, is quite good with over 90 % fairly successful estimations, which can be seen in tables 13 and 14. The accuracy in two dimensions is very good with almost 93 % completely successful estimations while in three dimensions the accuracy is a bit worse with almost 80 % completely successful estimations.

The average runtimes are quite long however leading to an update rate of around 19 Hz in two dimensions and 14 Hz in three dimensions. As stated in previous sections, this is greatly reduced (around a factor of ten) if input signals with fewer data points is used (as in the case of digital input signals). With an increased update rate, the impact of a low percentage of successful estimations is decreases.

The next part of this thesis is to implement the complete multiobject two-step procedure in an Android Application using Java.

8 Android Application

A large part of this master thesis is to create an application for Android using Java.

8.1 Introduction and Aim

The main purpose of the application is to be able to estimate distances using input data from a sensor prototype as well as estimate positions using the estimated distances and displaying the results to the user. The estimations are done using the complete two-step positioning algorithms found in the previous sections. The prototype is currently still in development and so the input data is generated in the prototype's microcontroller.

The microcontroller is connected to the Android smartphone via USB. This means that the application should be able to extract the input data via USB host mode on the phone.

The application should have two main pages (or Activities), the first one being the main page where the results are presented and the second being a settings page.

8.2 Graphical User Interface

On the main page there are three text views with three corresponding toggle buttons. The text views display the distance estimation, the position estimation in two dimensions and the position estimation in three dimensions. The corresponding buttons let the user turn the estimations on or off. There is also a text view displaying different important messages, such as if the sensor is connected or not. The GUI for the main page can be seen in figure 23 below.

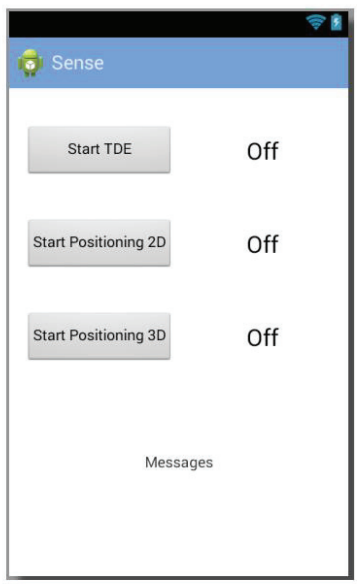


Figure 23. The main page of the application

In the settings page, different parameters such as gain or supply voltages can be set or changed as can be seen in figure 24 below.

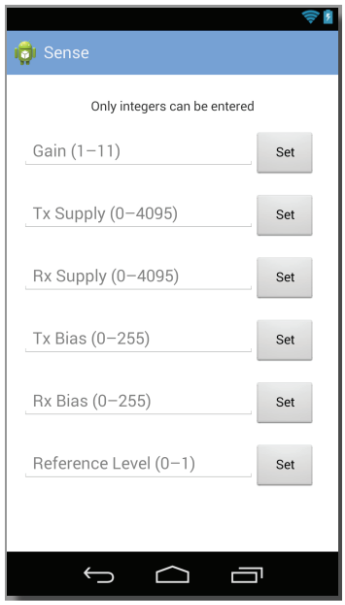


Figure 24. The Settings page of the application

8.3 General Implementation and Design

Since data should be retrieved via USB, the application should not just run with one thread. In addition to the main thread, which handles the GUI and responds to buttons being pressed and changes being made in the settings, additional worker threads should be used. The reason for this is that the retrieval of the input data from the microcontroller via USB might take time or in the worst case cause a timeout. If this happens in the main thread, the whole application seems slow or even dead.

When having more than one thread, however, synchronization is very important. For example if multiple threads have access to a setting's state, one thread may change the state of the setting while, at the same time, another thread is using the state. This can cause concurrency problems and in the worst case even give rise to deadlock.

In order to prevent this, a monitor handles all the shared data and strictly controls how and when the different threads are allowed to use this data.

In addition to the main thread there are two worker threads, one that handles the communication with the microcontroller and one that performs the distance as well as position estimation

algorithms. The algorithms implemented are the ones chosen previously in the thesis. All the Matlab functions such as findpeaks used in the Covariance algorithm and xcorr used when creating a reference are implemented “natively” in Java (i.e. no external library containing these functions is used).

Below, in figure 25, is the complete design of the application.

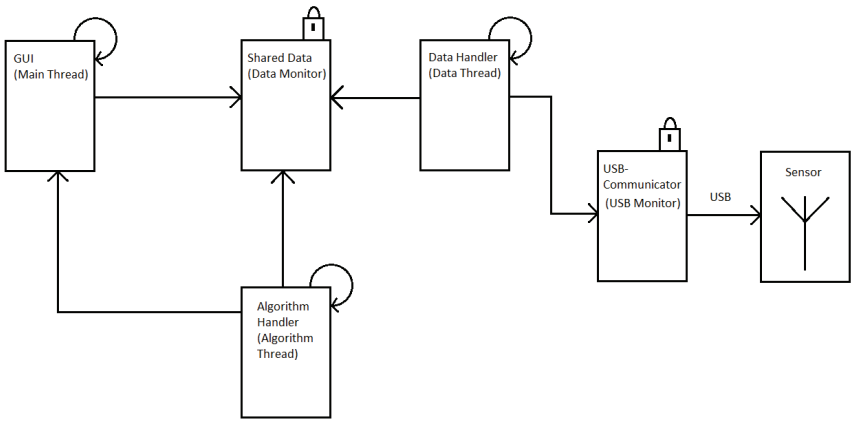


Figure 25. The design of the application only including monitors and threads

9 Conclusions

The goal of this thesis was to investigate algorithms for localizing multiple objects based on input from multiple wavelet radar sensors and then find the best, based on stability, accuracy and computational complexity, to implement in an Android application.

Based on all the test results above, one algorithm for time delay estimation (or distance estimation) and one algorithm for localization (positioning) were found to be the best suited for implementing in the Android application.

Regarding the time delay estimation; from the choice of the Peak Detection, Covariance and Least Squares algorithms, the Covariance algorithm proved to be superior in comparison to the other two. The main and most essential reason for this is its stability. When having either analog or digital input signals, this algorithm is able to find all reflections within a range of approximately 0.8 m, which is well above the supported range of 0.5 m.

Another thing to be investigated was how the sensors should be placed and how many that should be placed in order to minimize the position error, considering the placement constraints. When it came to the placement, the results were all very conclusive; the further apart the sensors are the better position accuracy. When it came to the sensor density, the results showed that the more sensors, the better accuracy. This is always true in two dimensions, but in three dimensions this is only true if the sensors are placed “properly”, meaning that they should not be linearly dependent (i.e. be placed on a line). However, there is a tradeoff between the number of sensors and the computational complexity. Also, the potential cost of more sensors should be considered. With all this taken into account, the number of sensors in two dimensions should be 3 and the number of sensors in three dimensions should be 4.

Considering the positioning; from the choice of the Linear Least Squares, Non-Linear Least Squares, Centroid and Weighted Centroid algorithms, the Weighted Centroid algorithm proved to be superior. The main reason for this is its very fast runtime and reasonable accuracy.

In the end a complete two-step positioning scheme (including the appurtenant algorithms) was found in order to estimate multiple objects with the following properties:

<u>In Two Dimensions:</u>

- Good stability with almost 94 % of the objects found within 1 cm of the true position, considering the large area where the objects were located
- Very good accuracy with almost 93 % of the objects found within 3 mm of the true position
- A fast runtime of 53.3 ms corresponding to an update rate of around 19 Hz for analog signals. This rate is greatly increased if digital signals are used

<u>In Three Dimensions:</u>

- Good stability with over 90 % of the objects found within 1 cm of the true position, considering the large area where the objects were located
- Good accuracy with almost 80 % of the objects found within 3 mm of the true position
- A fast runtime of 72.5 ms corresponding to an update rate of around 14 Hz for analog signals. This rate is greatly increased if digital signals are used

Finally the creation of the application, including implementing the complete two-step positioning scheme was in whole quite successful. Everything works seamlessly except the data retrieval via USB. For some reason the process suffers from heavy data loss when the data amount is larger than 300 bytes and done fast. The solution found for this is to obtain the data in smaller portions. This makes the transfer stable (i.e. no data loss) but in return takes a lot of time.

The input signals used in the application (i.e. the signals generated in the microcontroller) is digital signals of around 8 kilobytes. The process of acquiring these signals from the microcontroller can take as much as two seconds while the two-step positioning estimation has a runtime of approximately 15 ms (which corresponds to an update rate of around 67 Hz).

So in conclusion, a good two-step positioning scheme was found and successfully implemented in an Android application (with the exception of some USB connection issues).

References

Below, all the references used throughout this thesis are listed.

[1] Sadaphal, V. P. and Jain, B. (2005). Localization Accuracy and Threshold Network Density for Tracking Sensor Networks. *IEEE International Conference on Personal Wireless*

[2] Estrin, D., Girod, L., Pottie, G. and Srivastava, M. (2001). Instrumenting the World with Wireless Sensor Networks. *IEEE International Conference on Acoustics, Speech and Signal Processing,* Vol. 4, pp. 2033-2036.

[3] Evrendilek, C. and Akcan, H. (2011). On the Complexity of Trilateration with Noisy Range Measurements. *IEEE Communications Letters,* Vol. 15, pp. 1097-1099.

[4] Gezici, S. and Poor, H. V. (2009). Position Estimation via Ultra-Wide-Band Signals. *Proceedings of the IEEE,* Vol. 97, pp. 386-403.

[5] Weiss, A.J. (2004). Direct Position Determination of Narrowband Radio Frequency Transmitters. *IEEE Signal Processing Letters,* Vol. 11, pp. 513-516.

[6] Pourhomayoun, M. and Fowler, M.L. (2012). Sensor Network Distributed Computation for Direct Position Determination. *IEEE 7th Sensor Array and Multichannel Signal Processing Workshop,* pp. 125-128.

[7] Shih, C-Y. and Marrón, P. J. (2010). COLA: Complexity-Reduced Trilateration Approach for 3D Localization in Wireless Sensor Networks. *Fourth International Conference on Sensor Technologies and Applications,* pp. 24-32.

[8] Yu, Y., Wang, G., Li, Z. and Li, C. (2007). Alternating Combination Trilateration for Unknown Nodes of Sensor Networks. *IEEE International Conference on Control and Automation,* pp. 1747-1751.

[9] Gezici, S. (2007). A Survey on Wireless Position Estimation. *Wireless Personal Communications,* Vol. 44, pp. 263-282.

[10] Blom, G., Enger, J., Englund, G., Grandell, J. and Holst, L. (2005). Punktskattning. *Sannolikhetsteori och statistikteori med tillämpningar,* s. 253-263. Lund: Studentlitteratur.

[11] Li, J. and Wu, R. (1998). An Efficient Algorithm for Time Delay Estimation. *IEEE Transactions on Signal Processing,* Vol. 46, pp. 2231-2235.

[12] Caffery Jr, J. J. (2000). A New Approach to the Geometry of TOA Location. *IEEE Vehicular Technology Conference,* Vol. 4, pp. 1943-1949.

[13] Yu, Y-B. and Gan, J-Y. (2009). Self-Localization Using Alternating Combination Trilateration for Sensor Nodes. *International Conference on Machine Learning and Cybernetics,* Vol. 1, pp. 85-90.

[14] Björck, Å. (1996). Introduction. *Numerical Methods for Least Squares Problems,* s. 1-9. New York: Society for Industrial and Applied Mathematics.

Appendix A

All the raw data output from the Matlab tests is collected and presented in this appendix.

TDE Analog:

Number of times the reflection were not found for Peak Detection: 2111.000000
Number of times the reflection were not found for Covariance: 0.000000
Number of times the reflection were not found for Least Squares: 1.000000

Number of estimation errors larger than 0.2 mm for Peak Detection: 1098.000000
Number of estimation errors larger than 0.2 mm for Covariance: 20.000000
Number of estimation errors larger than 0.2 mm for Least Squares: 103.000000

The average runtime for Peak Detection is: 0.018571 s
The average runtime for Covariance is: 0.023367 s
The average runtime for Least Squares is: 0.021173 s

The std for Peak Detection is: 0.624188 mm
The std for Covariance is: 0.108795 mm
The std for Least Squares is: 0.146360 mm

TDE Digital:

Number of times the reflection were not found for Peak Detection: 343.000000
Number of times the reflection were not found for Covariance: 0.000000
Number of times the reflection were not found for Least Squares: 17.000000

Number of estimation errors larger than 3 mm for Peak Detection: 111.000000
Number of estimation errors larger than 3 mm for Covariance: 54.000000
Number of estimation errors larger than 3 mm for Least Squares: 37.000000

Number of estimation errors larger than 1 mm for Peak Detection: 204.000000
Number of estimation errors larger than 1 mm for Covariance: 172.000000
Number of estimation errors larger than 1 mm for Least Squares: 511.000000

The average runtime for Peak Detection is: 4.164115 ms
The average runtime for Covariance is: 6.590849 ms
The average runtime for Least Squares is: 6.604295 ms

The std for Peak Detection is: 78.341114 mm
The std for Covariance is: 2.832895 mm
The std for Least Squares is: 1.371314 mm

Position Estimation:

Two dimensions:

Results for std noise: 0.1 mm

Mean error for ML Gradient Descent: 0.426266 mm
Mean error for ML Brute Force: 0.94867 mm
Mean error for Weighted Centroid: 0.436449 mm
Mean error for Non-Weighted Centroid: 0.516982 mm

Number of positions not found for ML Gradient Descent: 0
Number of positions not found for ML Brute Force: 0
Number of positions not found for Weighted Centroid: 0
Number of positions not found for Non-Weighted Centroid: 0

Average runtime for ML Gradient Descent: 10.056 ms
Average runtime for ML Brute Force: 15.4133 ms
Average runtime for Weighted Centroid: 0.885082 ms
Average runtime for Non-Weighted Centroid: 0.593463 ms

Std for ML Gradient Descent: 0.589145 mm
Std for ML Brute Force: 1.24942 mm
Std for Weighted Centroid: 0.599785 mm
Std for Non-Weighted Centroid: 0.708657 mm

Results for std noise: 1 mm

Mean error for ML Gradient Descent: 4.37005 mm
Mean error for ML Brute Force: 4.35314 mm
Mean error for Weighted Centroid: 4.45542 mm
Mean error for Non-Weighted Centroid: 5.26482 mm

Number of positions not found for ML Gradient Descent: 0
Number of positions not found for ML Brute Force: 0
Number of positions not found for Weighted Centroid: 0
Number of positions not found for Non-Weighted Centroid: 0

Average runtime for ML Gradient Descent: 13.8901 ms
Average runtime for ML Brute Force: 15.0786 ms
Average runtime for Weighted Centroid: 0.931639 ms
Average runtime for Non-Weighted Centroid: 0.600766 ms

Std for ML Gradient Descent: 5.98913 mm
Std for ML Brute Force: 6.02142 mm
Std for Weighted Centroid: 6.0751 mm
Std for Non-Weighted Centroid: 7.16822 mm

Results for std noise: 5 mm

Mean error for ML Gradient Descent: 21.8711 mm
Mean error for ML Brute Force: 19.6694 mm
Mean error for Weighted Centroid: 23.0319 mm

Mean error for Non-Weighted Centroid: 28.7619 mm

Number of positions not found for ML Gradient Descent: 0
Number of positions not found for ML Brute Force: 0
Number of positions not found for Weighted Centroid: 0
Number of positions not found for Non-Weighted Centroid: 0

Average runtime for ML Gradient Descent: 17.8759 ms
Average runtime for ML Brute Force: 15.8789 ms
Average runtime for Weighted Centroid: 0.934035 ms
Average runtime for Non-Weighted Centroid: 0.629089 ms

Std for ML Gradient Descent: 30.5488 mm
Std for ML Brute Force: 27.5115 mm
Std for Weighted Centroid: 31.5073 mm
Std for Non-Weighted Centroid: 39.2913 mm

Results for std noise: 10 mm

Mean error for ML Gradient Descent: 42.2388 mm
Mean error for ML Brute Force: 35.1421 mm
Mean error for Weighted Centroid: 45.0942 mm
Mean error for Non-Weighted Centroid: 54.4973 mm

Number of positions not found for ML Gradient Descent: 0
Number of positions not found for ML Brute Force: 0
Number of positions not found for Weighted Centroid: 0
Number of positions not found for Non-Weighted Centroid: 0

Average runtime for ML Gradient Descent: 19.2498 ms
Average runtime for ML Brute Force: 15.5259 ms
Average runtime for Weighted Centroid: 0.924431 ms
Average runtime for Non-Weighted Centroid: 0.629535 ms

Std for ML Gradient Descent: 57.8063 mm
Std for ML Brute Force: 47.4401 mm
Std for Weighted Centroid: 59.6462 mm
Std for Non-Weighted Centroid: 71.1082 mm

Three dimensions

Results for std noise: 0.1 mm

Mean error for ML Gradient Descent: 0.683174 mm
Mean error for ML Brute Force: 5.19062 mm
Mean error for Weighted Centroid: 0.69029 mm
Mean error for Non-Weighted Centroid: 0.711684 mm

Number of positions not found for ML Gradient Descent: 0

Number of positions not found for ML Brute Force: 0
Number of positions not found for Weighted Centroid: 0
Number of positions not found for Non-Weighted Centroid: 0

Average runtime for ML Gradient Descent: 8.78941 ms
Average runtime for ML Brute Force: 30.7027 ms
Average runtime for Weighted Centroid: 0.445913 ms
Average runtime for Non-Weighted Centroid: 0.169061 ms

Std for ML Gradient Descent: 0.880172 mm
Std for ML Brute Force: 6.68011 mm
Std for Weighted Centroid: 0.884999 mm
Std for Non-Weighted Centroid: 0.910589 mm

Results for std noise: 1 mm

Mean error for ML Gradient Descent: 7.07616 mm
Mean error for ML Brute Force: 8.68799 mm
Mean error for Weighted Centroid: 7.12434 mm
Mean error for Non-Weighted Centroid: 7.35722 mm

Number of positions not found for ML Gradient Descent: 1
Number of positions not found for ML Brute Force: 0
Number of positions not found for Weighted Centroid: 1
Number of positions not found for Non-Weighted Centroid: 1

Average runtime for ML Gradient Descent: 17.079 ms
Average runtime for ML Brute Force: 31.7986 ms
Average runtime for Weighted Centroid: 0.45881 ms
Average runtime for Non-Weighted Centroid: 0.176094 ms

Std for ML Gradient Descent: 8.9505 mm
Std for ML Brute Force: 11.0076 mm
Std for Weighted Centroid: 9.02948 mm
Std for Non-Weighted Centroid: 9.37277 mm

Results for std noise: 5 mm

Mean error for ML Gradient Descent: 34.2459 mm
Mean error for ML Brute Force: 30.8797 mm
Mean error for Weighted Centroid: 34.341 mm
Mean error for Non-Weighted Centroid: 35.725 mm

Number of positions not found for ML Gradient Descent: 4
Number of positions not found for ML Brute Force: 0
Number of positions not found for Weighted Centroid: 4
Number of positions not found for Non-Weighted Centroid: 4

Average runtime for ML Gradient Descent: 22.308 ms

Average runtime for ML Brute Force: 33.1178 ms
Average runtime for Weighted Centroid: 0.492239 ms
Average runtime for Non-Weighted Centroid: 0.192077 ms

Std for ML Gradient Descent: 43.589 mm
Std for ML Brute Force: 38.7789 mm
Std for Weighted Centroid: 43.9149 mm
Std for Non-Weighted Centroid: 45.7803 mm

Results for std noise: 10 mm

Mean error for ML Gradient Descent: 68.7852 mm
Mean error for ML Brute Force: 53.209 mm
Mean error for Weighted Centroid: 69.2854 mm
Mean error for Non-Weighted Centroid: 71.9422 mm

Number of positions not found for ML Gradient Descent: 8
Number of positions not found for ML Brute Force: 0
Number of positions not found for Weighted Centroid: 8
Number of positions not found for Non-Weighted Centroid: 8

Average runtime for ML Gradient Descent: 30.487 ms
Average runtime for ML Brute Force: 38.0675 ms
Average runtime for Weighted Centroid: 0.564669 ms
Average runtime for Non-Weighted Centroid: 0.226602 ms

Std for ML Gradient Descent: 86.0775 mm
Std for ML Brute Force: 65.0071 mm
Std for Weighted Centroid: 86.6997 mm
Std for Non-Weighted Centroid: 90.3303 mm

Position estimation multiple objects:

Two dimensions:

2 Objects placed and 10000 iterations:

Average runtime: 1.84606 ms
Number of times no object was found: 0
Number of times an object was not found: 152
Number of times more than 2 objects was found: 199
Number of times the error was larger than 3 cm: 192
Percentage completely successful estimations: 96.13%
Percentage fairly successful estimations: 98.48%
Average position error: 5.92267 mm

3 Objects placed and 10000 iterations:

Average runtime: 4.10898 ms
Number of times no object was found: 0
Number of times an object was not found: 287
Number of times more than 3 objects was found: 795
Number of times the error was larger than 3 cm: 698
Percentage completely successful estimations: 88.54%
Percentage fairly successful estimations: 97.13%
Average position error: 9.96327 mm

4 Objects placed and 10000 iterations:

Average runtime: 7.64545 ms
Number of times no object was found: 0
Number of times an object was not found: 433
Number of times more than 4 objects was found: 1750
Number of times the error was larger than 3 cm: 1566
Percentage completely successful estimations: 77.12%
Percentage fairly successful estimations: 95.67%
Average position error: 17.6335 mm

5 Objects placed and 10000 iterations:

Average runtime: 12.5332 ms
Number of times no object was found: 0
Number of times an object was not found: 519
Number of times more than 5 objects was found: 2929
Number of times the error was larger than 3 cm: 2748
Percentage completely successful estimations: 63.7%
Percentage fairly successful estimations: 94.81%
Average position error: 25.4392 mm

Three Dimensions:

2 Objects placed and 10000 iterations:

Average runtime: 4.95478 ms
Number of times no object was found: 0
Number of times an object was not found: 58
Number of times more than 2 objects was found: 1216
Number of times the error was larger than 3 cm: 1154
Percentage completely successful estimations: 87.15%
Percentage fairly successful estimations: 99.42%
Average position error: 29.9996 mm

3 Objects placed and 10000 iterations:

Average runtime: 19.4841 ms
Number of times no object was found: 0
Number of times an object was not found: 98
Number of times more than 3 objects was found: 3489
Number of times the error was larger than 3 cm: 3334
Percentage completely successful estimations: 63.95%
Percentage fairly successful estimations: 99.02%
Average position error: 53.2048 mm

4 Objects placed and 10000 iterations:

Average runtime: 55.1599 ms
Number of times no object was found: 0
Number of times an object was not found: 94
Number of times more than 4 objects was found: 6004
Number of times the error was larger than 3 cm: 5769
Percentage completely successful estimations: 38.71%
Percentage fairly successful estimations: 99.06%
Average position error: 73.2099 mm

5 Objects placed and 10000 iterations:

Average runtime: 131.352 ms
Number of times no object was found: 0
Number of times an object was not found: 49
Number of times more than 5 objects was found: 8003
Number of times the error was larger than 3 cm: 7818
Percentage completely successful estimations: 19.14%
Percentage fairly successful estimations: 99.51%
Average position error: 92.806 mm

Two-step one object:

Two dimensions:

Mean error for ML Gradient Descent: 0.0860555 mm
Mean error for ML Brute Force: 0.588426 mm
Mean error for Weighted Centroid: 0.0844301 mm
Mean error for Non-Weighted Centroid: 0.0927672 mm

Number of times a distance was not found: 4
Number of positions not found for ML Gradient Descent: 4
Number of positions not found for ML Brute Force: 4
Number of positions not found for Weighted Centroid: 4
Number of positions not found for Non-Weighted Centroid: 4

Average runtime for TDE using Covariance: 95.3098 ms
Average runtime for ML Gradient Descent: 6.57147 ms
Average runtime for ML Brute Force: 44.8017 ms
Average runtime for Weighted Centroid: 1.24765 ms
Average runtime for Non-Weighted Centroid: 0.845018 ms

Std for ML Gradient Descent: 0.286488 mm
Std for ML Brute Force: 0.7878 mm
Std for Weighted Centroid: 0.224871 mm
Std for Non-Weighted Centroid: 0.165998 mm
Std for Covariance: 0.0432456 mm

Three dimensions:

Mean error for ML Gradient Descent: 0.115921 mm
Mean error for Weighted Centroid: 0.118677 mm
Mean error for Non-Weighted Centroid: 0.117067 mm

Number of times a distance was not found: 10
Number of positions not found for ML Gradient Descent: 10
Number of positions not found for Weighted Centroid: 10
Number of positions not found for Non-Weighted Centroid: 10

Average runtime for TDE using Covariance: 130.559 ms
Average runtime for ML Gradient Descent: 5.27132 ms
Average runtime for Weighted Centroid: 0.679709 ms
Average runtime for Non-Weighted Centroid: 0.229833 ms

Std for ML Gradient Descent: 0.141178 mm
Std for Weighted Centroid: 0.148015 mm
Std for Non-Weighted Centroid: 0.143537 mm
Std for Covariance: 0.0629875 mm

Two-step two objects:

Two dimensions:

2 Objects placed and 1000 iterations:

Average runtime TDE: 51.4385 ms
Average runtime positioning: 1.87006 ms
Number of times a distance was erroneously estimated: 71
Number of times no object was found: 0
Number of times an object was not found: 0
Number of times more than 2 objects was found: 0
Number of times an error was larger than 3 cm: 0
Percentage completely successful estimations: 92.9%
Percentage fairly successful estimations: 93.8%
Average distance error: 0.0506116 mm

Average position error: 0.107519 mm

Three dimensions:

2 Objects placed and 1000 iterations:

Average runtime TDE: 68.5043 ms
Average runtime positioning: 4.01215 ms
Number of times a distance was erroneously estimated: 123
Number of times no object was found: 0
Number of times an object was not found: 0
Number of times more than 2 objects was found: 95
Number of times an error was larger than 3 mm: 95
Percentage completely successful estimations: 78.2%
Percentage fairly successful estimations: 90.4%
Average distance error: 0.0505353 mm
Average position error: 33.415 mm

LUND
UNIVERSITY

http://www.eit.lth.se