Saarland University
Faculty of Natural Sciences and Technology I
Department of Computer- and Communications Technology

Experimental study of object tracking and its applications

Bachelor's Thesis

submitted by
Dominic Buchheit

submitted on
November 9th, 2016

Supervisor and Advisor
Prof. Dr. Dietrich Klakow
Dipl. Ing. Youssef Oualil

Reviewers
Prof. Dr. Dietrich Klakow
Prof. Dr. Bernd Finkbeiner

Eidesstattliche Erklärung
Ich erkläre hiermit an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Statement in Lieu of an Oath
I hereby confirm that I have written this thesis on my own and that I have not used any other media or materials than the ones referred to in this thesis.

Einverständniserklärung
Ich bin damit einverstanden, dass meine (bestandene) Arbeit in beiden Versionen in die Bibliothek der Informatik aufgenommen und damit veröffentlicht wird.

Declaration of Consent
I agree to make both versions of my thesis (with passing grad) accessible to the public by having them added to the library of the Computer Science Department.

Saarbrücken, _____ _____
(Datum/Date) (Dominic Buchheit)

Acknowledgments

Firstly, I thank my supervisor, Professor Dr. Dietrich Klakow for the continuous support during the preparation and the thesis itself.

Besides my supervisor, I extend my thanks to my advisor Dipl. Ing. Youssef Oualil for his motivation, immense patience and knowledge I experienced during the entire time. Without his support it would not have been possible to write this thesis.

My sincere thanks also go to my second assessor Prof. Dr. Bernd Finkbeiner.

Many thanks go to the members of the LSV and the chair itself for giving me feedback during the preparation phase and providing information as well as the infrastructure needed to work on this thesis.

Abstract

In current times, object localization and tracking is getting more and more important. There are different approaches with individual advantages and flaws, mostly contradictory. Low error rates usually mean high computational effort, while low computational effort means high error rates. Existing algorithms try to find a reasonable exchange between these two extremes. The object of this work is to implement some of the most common localization and tracking algorithms and study their behavior in different scenarios using different configurations.

Contents

1 Introduction

The following chapters describe the motivation and define some basic terms needed in the course of this thesis.

1.1 Motivation

As described in the seminar paper [7], there are several different algorithms and methods to localize and/or track generic objects in their environment. Some of them were developed for a special case, but most of them are generic for any kind of *measurement-object* combination. For example, a quantity to measure (and to localize/track) could be a *temperature* as well as an *image* or an *audio signal*. On the other side, an object to track could be a *person*, a *robot* or a *thing* in general. As mentioned in the seminar paper [7], the focus of this work lays on tracking speaking persons. Each of the algorithms -so called *filters*- has its advantages and disadvantages, some are more robust than others, depending on the scenario and parameters set.

1.2 Time Difference Of Arrival

We assume having n microphones ($\{m_1, m_2, ..., m_n\}$) at pairwise disjunct positions in a space. The *Time Difference Of Arrival* (**TDOA**) describes the delay (in time units) between an arriving signal at microphone m_j with respect to the same signal at microphone m_i. *Same signal* in this case means the audio waves produced by a speaker at an unknown position, sampled by m_i and m_j, respectively. Sometimes, authors refer to *Sample Difference Of Arrival* (SDOA) instead of TDOA. The only difference is that TDOA respects the (constant) sample frequency f of the sensors while SDOA does not. It follows the identity:

$$TDOA = \frac{SDOA}{f} \tag{1}$$

Obviously, *TDOA* and *SDOA* are directly proportional ($SDOA \sim TDOA$). Therefore, if we talk about *TDOA*, it is also valid for *SDOA* and vice versa. Using the far-field-assumption[1], the TDOA of the signal produced by object x between the microphones m_i and m_j can be defined as described in **equation 2** [2]:

$$\tau_{m_i,m_j}(x) = \frac{||x - m_i|| - ||x - m_j||}{s} \tag{2}$$

where s is the speed of sound. A similar approach as used in [3], uses, again under the far-field-assumption, the angle of arrival instead of the absolute position x of the object to track:

$$\tau_{m_i,m_j}(\Theta, \Phi) = \frac{d[\Theta, \Phi]^T \cdot (m_i - m_j)}{s} \tag{3}$$

[1] From a certain distance to the speaker on the wave-front of an audio signal could be viewed as a straight line (instead of a curved line)

where $d[\Theta, \Phi] = [\cos(\Phi)\sin(\Theta), \cos(\Phi)\cos(\Theta), \sin(\Phi)]$ denotes the spherical-to-Cartesian coordinate transformation of the angle of arrival, assuming a distance to the speaker of $r = 1m$.

Figure 1 shows a schematic description of a TDOA scenario in 2-D space, where x is the object of interest, m_i and m_j are the audio sensors (microphones), a is the distance of the wave front already moved and d is the wave-front difference between m_j and m_i. In this case the TDOA is obviously $\tau_{m_i,m_j}(x) = \frac{d}{s}$. In

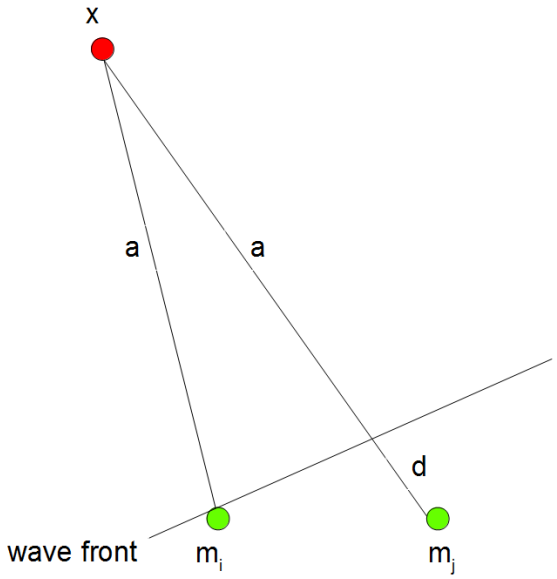

Figure 1: *Schematic description of TDOA scenario.*

practice, most of the time, the delay $\tau_{m_i,m_j}(x)$ is measurable, while the distance between x and the microphones m_i and m_j, respectively, as well as the azimuth[2], are the (unknown) objects of interest.

1.3 Generalized Cross Correlation With Phase Transform

The Generalized Cross Correlation with Phase Transform (**GCC-PHAT**) provides a TDOA estimation of a signal between two microphones. Given two signals $x_i(n)$ and $x_j(n)$ of the two microphones i and j, GCC is defined as

$$R_q(\tau) = \frac{1}{2\pi}\int_0^{2\pi} \Phi(\omega)X_i(\omega)(X_j(\omega))^* e^{j\omega\tau} d\omega \tag{4}$$

where $(\cdot)^*$ denotes the complex conjugate, and $X_i(\omega)$ and $X_j(\omega)$ are the Fourier transform of both signals x_i and x_j, respectively. $\Phi(\omega)$ denotes a pre-filter. A

[2]The angle between x and the sensors

common choice of $\Phi(\omega)$ is the phase transform (PHAT) weighting:

$$\Phi(\omega) = \frac{1}{|X_i(\omega)(X_j(\omega))^*|} \tag{5}$$

1.4 Steered Response Power - Phase Transform

The Steered Response Power (**SRP**) is probably the easiest way to *localize* an object of interest. While *easiest* refers to the simplicity, the algorithm is highly computationally inefficient. The steered response returned from a certain source location q can be calculated as [4]

$$SRP(q) = 4\pi \sum_{m=1}^{M} (R_m(\tau_m(q))) + \mathcal{K} \tag{6}$$

where M denotes the *number of microphone pairs* and $\mathcal{K}$ is introduced by the auto-correlation of each microphone pair (see [4] for more details). Since $\mathcal{K}$ is constant, we ignore it during the rest of this thesis.

In order to find the source $\hat{q}$ of the signal, one tries to maximize $SRP(q)$ over all positions Q in the 2-D/3-D space:

$$\hat{q} = \arg\max_{q \in Q} SRP(q) \tag{7}$$

In other words: SRP iterates over a defined grid (all positions in the scene), in order to find the state which is *most likely* the source position of the object. Thus, the computationally complexity is in order of $O(n^2 m)$, where n is the number of microphones and m is the number of states in the grid.
Figure 2 shows the basic qualitative tool chain of the *SRP localization* algorithm. When talking about SRP with Phase Transform (**SRP-PHAT**), we imply a PHAT weighted GCC estimation $R_m(\tau_m(q))$ under SRP.

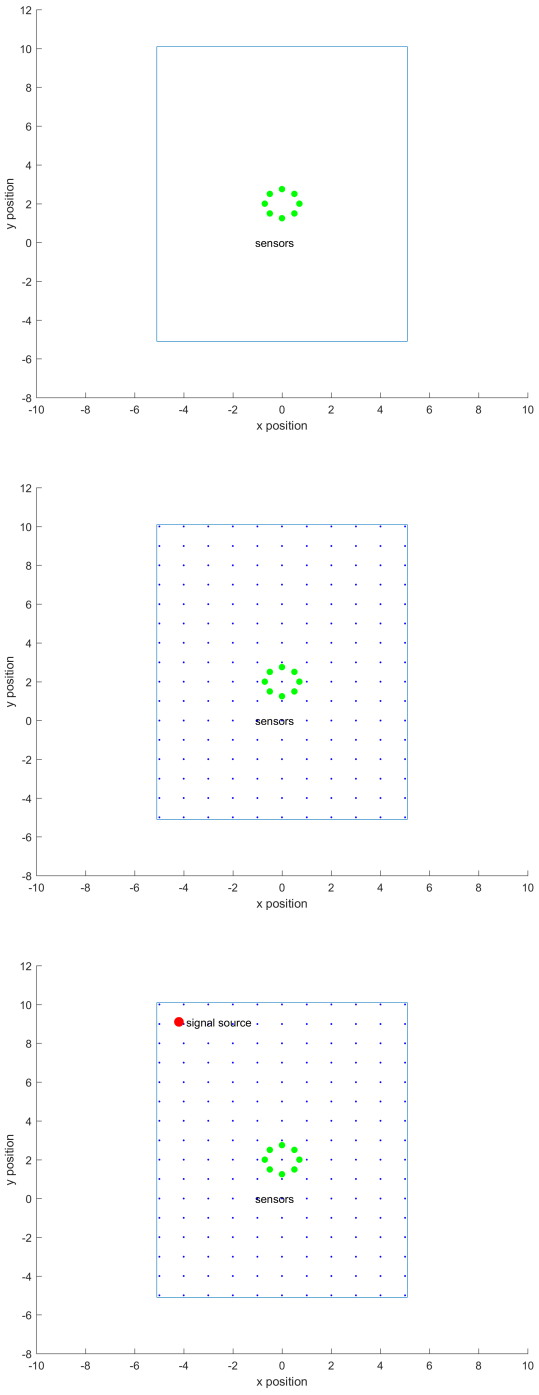

Figure 2: Top: *Define room and sensor positions.* ***Middle:*** *Superimpose a grid net.* ***Bottom:*** *Apply SRP on grid.*

2 Thesis Description

Goal of this thesis is to implement and use several algorithms for object tracking, in order to find differences in their behavior in certain scenarios. As mentioned, the focus lays on audio signals produced by speakers. These signals were recorded by a number of microphones, located in a relatively small room. The room dimensions, the locations of the microphones and the speaker's movement will be described in more details in the following chapters.
Roughly, the thesis describes the following steps:

1. Implement different algorithms for object tracking.
2. Define scenarios and performance standards.
3. Simulate/build the scenarios.
4. Run a performance study using the implemented filters on top of the scenarios.

3 Thesis Environment

During the thesis, we defined the environment where we performed all the experiments. This environment will be defined in the following sub-chapters.

3.1 Room

In order to provide a consistent basis for each filter and scenario, we decided to simulate audio signals rather than to record them. A further benefit of an approach like this is that one can easily adapt the framework conditions by adding more or less (or none) noise, as well as different speaker movements. The room is defined as follows:

- Dimensions: $5m \times 3m \times 2.5m$
- One microphone array with 8 microphones.
- Microphone array is located near the center at $(2.5m/1.5m/0.9m)$, uniformly arranged on a circle with radius $0.06m$.
- Carpeted floor, sound absorbing material on the ceiling.

We used the MATLAB implementation of the *fastISM* algorithm by Eric Lehmann [8], based on [13] and [15], to simulate the room as described above.

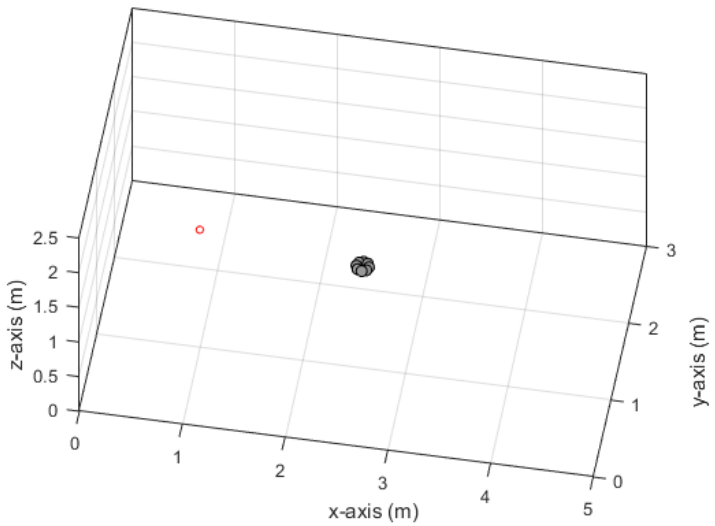

Figure 3: *Room containing a microphone array (gray) and an exemplary speaker (red).*

Figure 3 exemplary shows the room dimensions.

3.2 Speaker Movement

Besides the *static speaker scenario* as described in **figure 3**, we defined several other *moving speaker scenarios*:

1. straight line
2. straight line with an edge
3. nonlinear line with an edge
4. nonlinear line
5. straight lines but with a sudden appearance at another position
6. circle around the microphone array

For the experiments itself we used the *static* scenario and *4 to 6*.
For each of the scenarios, we assumed a static velocity of the speaker. Otherwise, it would be mandatory to track the speaker's velocity as well, which would enormously complicate the scenarios. Since we are mainly interested in the direction to the speaker (relative to the center of the microphone), we also defined the speakers height to $1.7m$.
Figure 4 shows a plot of the scenarios from the bird's eye perspective. In every scenario, except of the circle movement, the speaker starts at the lower left corner and moves along the red marked trajectory.

3.3 Noise

The *fastISM* implementation allows to add noise in a definable level relative to the signal, the so called signal-to-noise ratio (**SNR**). Generally speaking, the SNR is defined as the relation between the *mean power* of the *useful signal* and the *mean power* of the added *noise*. The mean power of a (real) signal s at time t is defined as:

$$\hat{P}(t) = \lim_{T\to 0} \frac{1}{2T} \int_{-T}^{T} s(t) \cdot s^*(t) dt \overset{s(t) \text{ is real}}{=} \lim_{T\to 0} \frac{1}{2T} \int_{-T}^{T} s(t)^2 dt \tag{8}$$

Using **equation 8**, the generic SNR is defined as follows:

$$SNR_{generic} = \frac{P_{useful\_signal}}{P_{noise}} \tag{9}$$

Because of the fact that in most scenarios the noise is much lower than the useful signal ($\Rightarrow P_{noise} \ll P_{useful\_signal}$), the SNR often gets defined logarithmically and is measured in the pseudo-unit decibel (*dB*):

$$SNR = 10 \cdot log(\frac{P_{useful\_signal}}{P_{noise}}) dB \tag{10}$$

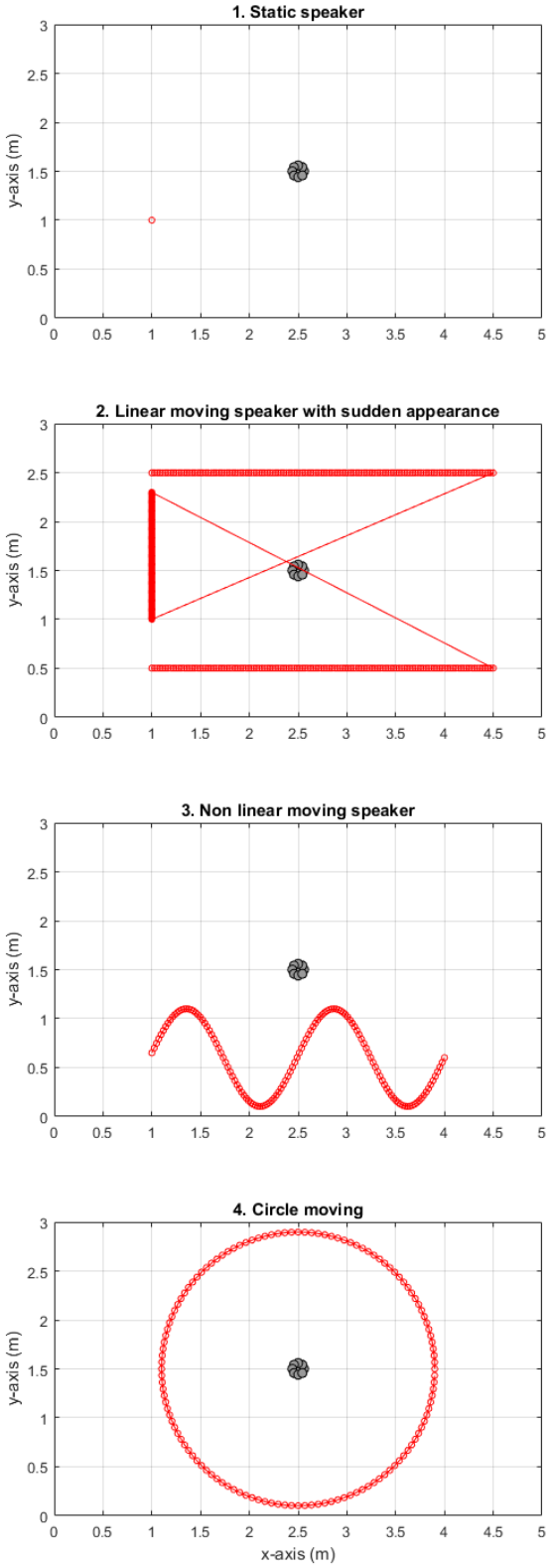

Figure 4: 1. *Static speaker.* ***2.*** *Linear movement with sudden appearance at another position.* ***3.*** *Non linear movement.* ***4.*** *Circular movement.*
Bird's eye perspective, speaker (red) at height $z = 1.7m$*, microphone at* $z = 0.9m$*, the z-axis stretches out of the paper layer.*

SNR value range (dB)	Quality of signal
40 - 20	very good
19 - 10	good/okay
9 - 5	bad
4 - 0	very bad
-1 - -12	not usable

Table 1: *Typical classification of signal-to-noise ratios.*

FastISM uses the latter definition and a white Gaussian noise. **Table 1** shows a typical classification of signal-to-noise ratios.
We used different levels of SNR for each scenario: A *low noise* level ($30dB$), a *medium noise* level ($15dB$) and a *heavy noise* level ($5dB$).

3.4 Scenario Outputs

After defining, configuring and simulating each of the scenarios, fastISM produces **multi channel audio signals**. Each channel represents a real discrete signal, sampled by a single microphone. In our case, we defined a microphone array containing 8 microphones resulting in a 8 channel audio file.
FastISM creates these audio files by simulating the Room Impulse Response (**RIR**) and coat over a configurable audio file.

Besides the audio files, we used the defined speaker trajectory to calculate the speaker's real positions, the so called **ground truth**. The ground truth consists of the speaker's Cartesian coordinates and later gets uniformly stretched to fit the duration of the audio file. This approach is valid since we assumed the speaker's moving velocity as constant.
Having the ground truth in Cartesian coordinates has several advantages: The biggest one is that we know the real initial position of the speaker and we are able to transform this position to spherical coordinates at any time to get the signal's direction of arrival.
Figure 5 shows the plotted samples of an audio file. The *colorful* peaks in the background lead to the conclusion that the channels are in fact delayed since otherwise the plots would fit perfectly over each other. The right side of **figure 5** shows a plot of the ground truth in form of the angle of arrival over the time in samples.

3.5 Limitations Of Test Data

Our simulation assumes ideal environment conditions, there is **no unexpected additional noise** besides the noise produced by the speaker and/or the microphones. This means that unexpected situations like a blowing light bulb or a projector's fan are not considered.

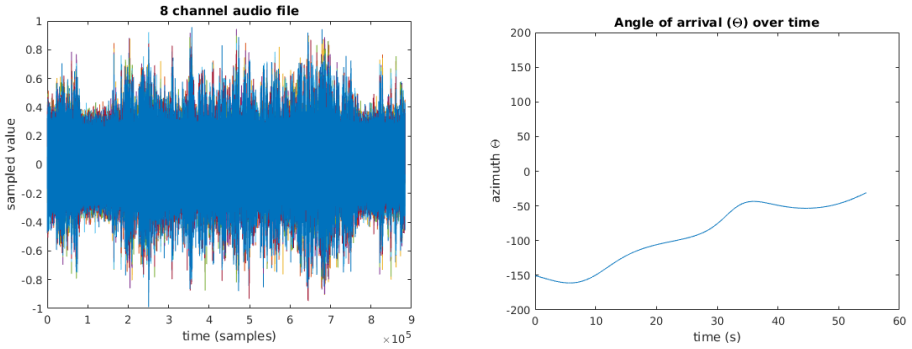

Figure 5: Left: *8 channel audio file plotted over each other.* ***Right:*** *Ground truth of a nonlinear moving speaker (**figure 4**, picture 3).*

As mentioned in **chapter 3.4**, the simulation needs an audio file to produce the microphone signals. We selected a file where the **speaker speaks without any interruption**, in order to make sure that there are always samples available for tracking and the filters are getting disturbed as less as possible. However, as part of this thesis, we wanted to check the filters behavior with missing samples and/or longer intervals without a talking speaker. Therefore, we created additional audio files for the simulation, based on the first file, and **added some silence periods** where the pure audio signal equals zero.

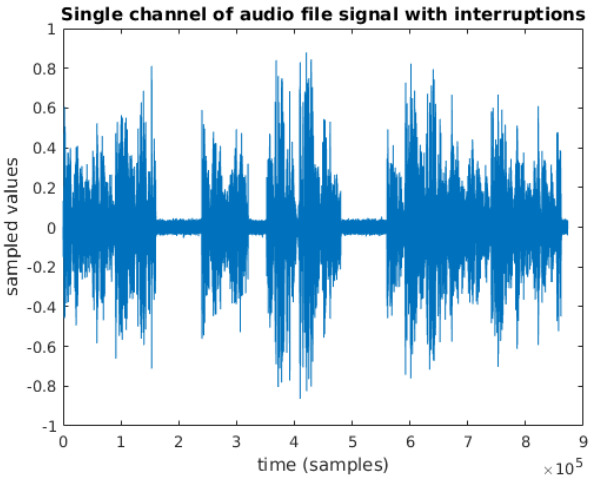

Figure 6: *A single channel of a simulated audio signal with 8 channels, containing three silence periods, two major and one minor, SNR:* $20dB$.

Figure 6 shows a single channel of the simulated audio file. Note that, due to the added noise, the signal during the silence periods is not zero at all. In the case of **figure 6** we used a fairly good signal-to-noise ratio of $20dB$. All the scenarios are designed for one single speaker. There is no multi-speaker support in this thesis.

4 Tracking Algorithms

This chapter explains the principle behind filters in general and introduces the filters used in this thesis.

4.1 A General Filter

Unlike pure localization algorithms such as *SRP-PHAT* (**chapter 1.4**), tracking filters should be able to **track** an object (online) and **predict** its next state. Most of the filters follow a three point plan as illustrated in **figure 7**.

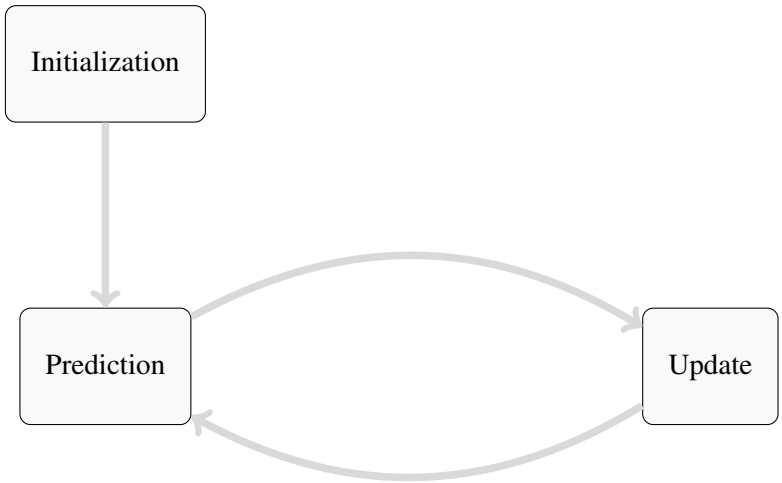

Figure 7: *Three point plan for filters.*

Algorithm 1 provides a pseudo code implementation of filters in general. All of the filters used in this thesis are following this concept.

1 *Initialize* the filter's individual parameters
2 **For each** iteration (i. e. each incoming measurement)
3 *Predict* the *object's state* on base of the last prediction
4 *Predict* the *object's next measurement* based on the predicted state
5 *Update the prediction* using the incoming measurement
6 **End for each**

Algorithm 1: Generic filter in pseudo code

Annotation: Our used MATLAB implementations of all the following filters can be found in the appendix.

4.2 Variable Explanations

With this chapter, we want to clarify variables and terms often used in the following chapters.

4.2.1 Object State

An object state x_k describes the *object of interest* x at a *certain time step* k. An object of interest could be for example a *position of a robot in the room* or a *filling level of a hydroelectric plant*. Let $\mathcal{X}$ be the set of all possible states (e. g. all positions in the room), then $x \in \mathcal{X}$ describes a single object state. $\mathcal{X}_{0:k}$ describes the object's states from time 0 to k.
$\hat{x}_k$ means an estimation of the object state x at time k.

4.2.2 Measurement

Similar to the object state, a measurement y_k at time k can have different meanings (position, filling, time delays). We differ between *direct measurements*, where the object's state gets measured without the need of any transformation, and measurements that need some kind of *transformation*. In other words: If y is in the same space as x than y is considered as direct measurement, while for any transition function h so that $h(x) = y \in \mathcal{Y}$, y is considered as indirect measurement of the state x. $\mathcal{Y}$ describes the set of all possible measurements and $\mathcal{Y}_{0:k}$ describes the measurements from time 0 to k.
$\hat{y}_k$ means an estimation of the measurement y at time k.

4.2.3 The Object State Transition Function

The object state transition function f describes the relation between the object state x_k at time k and its predecessor x_{k-1} at time $k-1$. It is needed to predict the object's next state, respecting some additional noise v_k. It can be a nonlinear function so that

$$x_k = f(x_{k-1}) + v_k. \tag{11}$$

If f is a linear system of equations, one could represent it in its matrix representation F so that

$$x_k = F \cdot x_{k-1} + v_k. \tag{12}$$

4.2.4 The Measurement Transition Function

The measurement transition function h maps an object state x_k at time k to the measurement y_k produced by this state, respecting some additional noise w_k. It can be nonlinear so that

$$y_k = h(x_k) + w_k. \tag{13}$$

If f is a linear system of equations, one could represent it in its matrix representation H so that

$$y_k = H \cdot x_{k-1} + w_k. \tag{14}$$

4.2.5 The Object State Covariance Matrix

P_k is called covariance matrix of the state at time k. It basically describes how reliable the current estimation of the object state is. It is needed to form the knowledge about the object state at a certain time, together with the estimated object state:

$$x_k \sim \mathcal{N}(\hat{x}_k, P_k) \tag{15}$$

where x_k describes the object state and follows a normal distribution ($\mathcal{N}$) with an object state estimation $\hat{x}_k$ as mean and the P_k as covariance.

4.2.6 Noise Covariance Matrices

Similar to the state covariance matrix, which describes the reliability, the noise covariance matrices describe how reliable the state model transition function and the measurement transition function are. Having a signal-to-noise ratio (*SNR*) of a white noise, determining the measurement covariance matrix R is simple:

$$R = I \cdot \frac{1}{10^{\frac{SNR}{10}}} \tag{16}$$

where I is the identity matrix and SNR is the signal to noise ratio in dB. We use the white uncorrelated Gaussian noises v_k and w_k and their covariances Q_k and R_k in order to express the noise added to the transition functions at time k.

4.3 Assumptions

Single object tracking is meant to be concerned with the tracking process of *one single object*. The object itself could either be *static* in its current state or moving on a *linear*, *nonlinear* or even *random* trajectory. The following algorithms are all capable of tracking one object, mostly using the following assumptions:

A1 Object dynamics ($x_{k-1} \rightarrow x_k$) and measurement equations (y_k) are linear:

$$\begin{aligned} x_k &= Fx_{k-1} + v_k \\ y_k &= Hx_k + w_k \end{aligned} \tag{17}$$

A2 v_k and w_k are zero mean white uncorrelated Gaussian noise sequences.

A3 The posterior density of the object state $p(x_{k-1}|y^{k-1})$ at time $k-1$ is Gaussian with mean x_{k-1} (latest state estimation).

4.3.1 Kalman Filter

Since the Kalman Filter (**KF**) relies on every single one of the three assumptions **A1-A3**, its implementation is simple, compared to the other filters in this document. Challa et al. [1] derived the filter as in **algorithm 2**.

1: Prediction (predicted mean and covariance matrix):

$$\begin{aligned} \hat{x}_{k|k-1} &= F\hat{x}_{k-1|k-1} \\ P_{k|k-1} &= FP_{k-1|k-1}F^T + Q_k \end{aligned} \tag{18}$$

2: Predicted measurement, innovation covariance matrix and Kalman gain:

$$\begin{aligned} \hat{y}_{k|k-1} &= H\hat{x}_{k|k-1}, \\ S_k &= HP_{k|k-1}H^T + R_k, \\ K_k &= P_{k|k-1}H^T S_k^{-1} \end{aligned} \tag{19}$$

3: Posterior mean (new position) and covariance matrix

$$\begin{aligned} \hat{x}_{k|k} &= \hat{x}_{k|k-1} + K_k(y_k - \hat{y}_{k|k-1}), \\ P_{k|k} &= P_{k|k-1} - K_k H_k P_{k|k-1} \end{aligned} \tag{20}$$

Algorithm 2: Kalman Filter.

The Kalman Filter nicely shows the two step technique as described in **figure 7**. While steps 1 and 2 predict the objects state and the next measurement, respectively, the third step corrects these predictions based on the actual input. The state covariance matrix ($P_{k|k-1}$) describes how reliable the prediction $\hat{x}_{k|k-1}$ is, based on the last state covariance matrix $P_{k-1|k-1}$ and the state transition matrix F, adding some transition noise using its covariance matrix Q_k.

As simple the Kalman Filter is, as limited it is, since it hardly relies on the mentioned assumptions (**chapter 4.3**). The biggest flaw of the Kalman Filter is the dependency of linear object and measurement transition functions, which excludes the Kalman Filter from being a candidate for the TDOA approach.

4.3.2 Extended Kalman Filter

The Extended Kalman Filter (**EKF**) only relies on the assumptions **A2** and **A3** and does not need linear models. In general, the EKF tries to approximate its nonlinear models in each iteration. Again, we implemented the EKF derived and proposed by Subhash Challa et al. (**algorithm 3**,[1]).

A question one may ask is: *"What happens if the transition functions are already linear?"*. This special case is indeed interesting, since then, it is not necessary to calculate the linearization. In other words, the EKF reduces to the normal KF what makes the KF a special case of the EKF.

1: Compute a *linearized approximation* of the *object state transition*:

$$F_k = \nabla_{x^T} f(x)|_{x=\hat{x}_{k-1|k-1}} \tag{21}$$

2: Predict the *object state* and *covariance matrix*:

$$\begin{aligned} \hat{x}_{k|k-1} &= f(\hat{x}_{k-1|k-1}), \\ P_{k|k-1} &= F_k P_{k-1|k-1} F_k^T + Q_k \end{aligned} \tag{22}$$

3: Compute a *linearized approximation* of the *measurement transition*:

$$H_k = \nabla_{x^T} h(x)|_{x=\hat{x}_{k|k-1}} \tag{23}$$

4: Predict the *measurement* and calculate the *innovation covariance* matrix and *Kalman Gain*:

$$\begin{gathered} \hat{y}_{k|k-1} = h(x_{k|k-1}), \qquad S_k = H_k P_{k|k-1} H_k^T + R_k, \\ K_k = P_{k|k-1} H_k^T S_k^{-1} \end{gathered} \tag{24}$$

5: Compute the *posterior object* state and *covariance matrix*:

$$\begin{aligned} \hat{x}_{k|k} &= \hat{x}_{k|k-1} + K_k(y_k - \hat{y}_{k|k-1}), \\ P_{k|k} &= P_{k|k-1} - K_k H_k P_{k|k-1} \end{aligned} \tag{25}$$

Algorithm 3: *Extended Kalman Filter, differences to the KF are marked red.*

4.3.3 Unscented Kalman Filter

The Unscented Kalman Filter (**UKF**) is an alternative to the EKF and therefore relies on the same assumptions. While it shares the computational simplicity with the EKF, it is easier to implement. The difference between this filter and the filters before is that there is no need to compute Hessian or Jacobian matrices. Instead, the filter uses the so called *unscented transformation* and *sigma points* to *estimate* the linearizations. **Figure 8** schematically shows the UT besides the linearization and the actual object state. [9] and [10] provide detailed information about the unscented transformation and about the parameters needed, as well as the number of sigma points needed. During the implementation we used their definition.

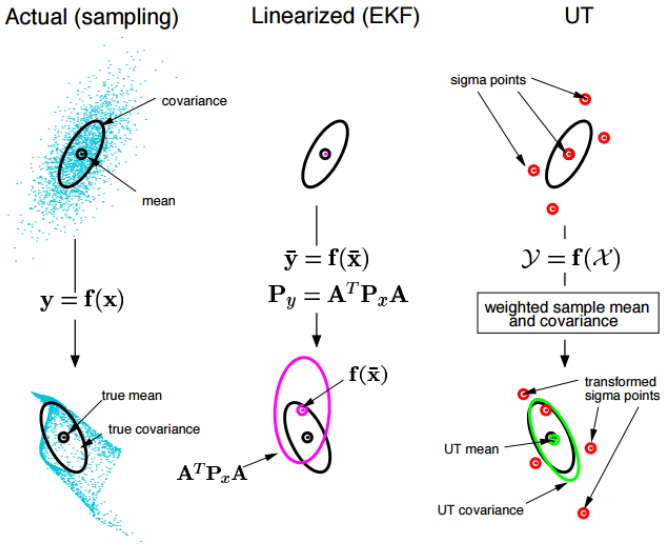

Figure 8: *Example of the Unscented Transformation for mean an covariance propagation.* ***Left:*** *actual.* ***Middle:*** *first order linearization (EKF).* ***Right:*** *Unscented Transformation. [9]*

Algorithm 4 again describes the implementation derived and proposed by Challa et al. [1]. Another good introduction is [14], provided by Julier et al.

1: Determine sigma points $X_{k-1}^1, ..., X_{k-1}^s$ and weights $w^1, ..., w^s$ to match a mean $\hat{x}_{k-1|k-1}$ and covariance matrix $P_{k-1|k-1}$

2: Compute the transformed sigma points $X_k^i = f(X_{k-1}^i), i = 1, ...s.$

3: Compute the predicted state statistics:

$$\begin{aligned} \hat{x}_{k|k-1} &= \sum_{i=1}^{s} w^i X_k^i \\ P_{k|k-1} &= Q_k + \sum_{i=1}^{s} w^i (X_k^i - \hat{x}_{k|k-1})(X_k^i - \hat{x}_{k|k-1})^T \end{aligned} \tag{26}$$

4: Determine sigma points $X_k^1, ..., X_k^s$ and weights $w^1, ..., w^s$ to match a mean $\hat{x}_{k|k-1}$ and covariance matrix $P_{k|k-1}$

5: Compute the transformed sigma points $Y_k^i = h(X_k^i), i = 1, ..., s.$

6: Compute the predicted measurement statistics:

$$\begin{aligned} \hat{y}_{k|k-1} &= \sum_{i=1}^{s} w^i Y_k^i \\ S_k &= R_k + \sum_{i=1}^{s} w^i (Y_k^i - \hat{y}_{k|k-1})(Y_k^i - \hat{y}_{k|k-1})^T \\ \Psi_k &= \sum_{i=1}^{s} w^i (X_k^i - \hat{y}_{k|k-1})(Y_k^i - \hat{y}_{k|k-1})^T \end{aligned} \tag{27}$$

7: Compute the posterior mean and covariance matrix:

$$\begin{aligned} \hat{x}_{k|k} &= \hat{x}_{k|k-1} + \Psi_k S_k^{-1} (y_k - \hat{y}_{k|k-1}) \\ P_{k|k} &= P_{k|k-1} - \Psi_k S_k^{-1} \Psi_k^T. \end{aligned} \tag{28}$$

Algorithm 4: Unscented Kalman Filter.

4.3.4 Particle Filter

In contrast to the filters before, the Particle Filter (**PF**) seeks a discrete approximation (stochastically) to the posterior probability density function. This has several advantages:

- The implementation is much simpler: no need to derive rules for determining the grid points, nor linearizing the models.
- The sampling procedure will automatically move sample points to the region of the state space of interest.
- Computational complexity is decreased (because the error convergence rate is independent of the dimension of the state).

Mostly, this filter is formualted as a Sequential Monte Carlo (**SMC**) method and uses so called *particles* to estimate the object's state. In general a particle filter has the form as described by Challa et al. (**figure 5**, [1]): For each iteration, there exists a number of particles n. If it is the first iteration (and there are not yet any particles), the particles are created (**A**) around the initial object state respecting a certain variance σ.
For each of these particles (= a possible object state), the filter now computes the posterior object's state. In other words: For each particle, it applies the object state transition function f as well as the measurement transition function h with respect to the noise covariances.

1: **for** $i = 1, ..., n$ **do**

2: Draw samples$(x_k^i, t^i) \sim x_0$

3: Compute the weight update factor:

$$e_k^i = \frac{p(y_k|x_k, y_{1:k-1})p(x_k|x_{k-1}^{t^i}), y_{1:k-1}}{q(x_k^i, t^i)} \tag{29}$$

4: **end for**

5: Compute the updated weights:

$$w_k^i = \frac{w_{k-1}^i e_k^i}{\sum_{j=1}^n w_{k-1}^j e_k^j}, i = 1, ..., n \tag{30}$$

Algorithm 5: Basic Particle Filter.

Using the transformed particles $\hat{y}_n$ and the actual measurement y the filter scores each of the particles and weights them (**B**). Finally, after resampling (**C**) the particles with respect to the weights, the posterior object's state gets estimated (**D**).

For each of the steps there exist different approaches. We used the following:

A Gaussian distribution with variance σ.

B Multivariate Gaussian distribution at y with mean $\hat{y}_n$ and the noise covariance matrices.

C Gaussian distribution with variance σ around the *best particles* (particles with highest weights).

D The mean of the resampled particles.

Obviously, the more particles used, the more accurate the filter will be as well as the execution time will increase. **Figure 9** shows schematically how the Particle Filter works, assuming the same domain of the object states and measurements. There are several other (improved) implementations such as the *Bootstrap filter for single-object tracking* or the *Optimal Importance Density for single-object tracking*. Both (and more) are described and derived in [1].

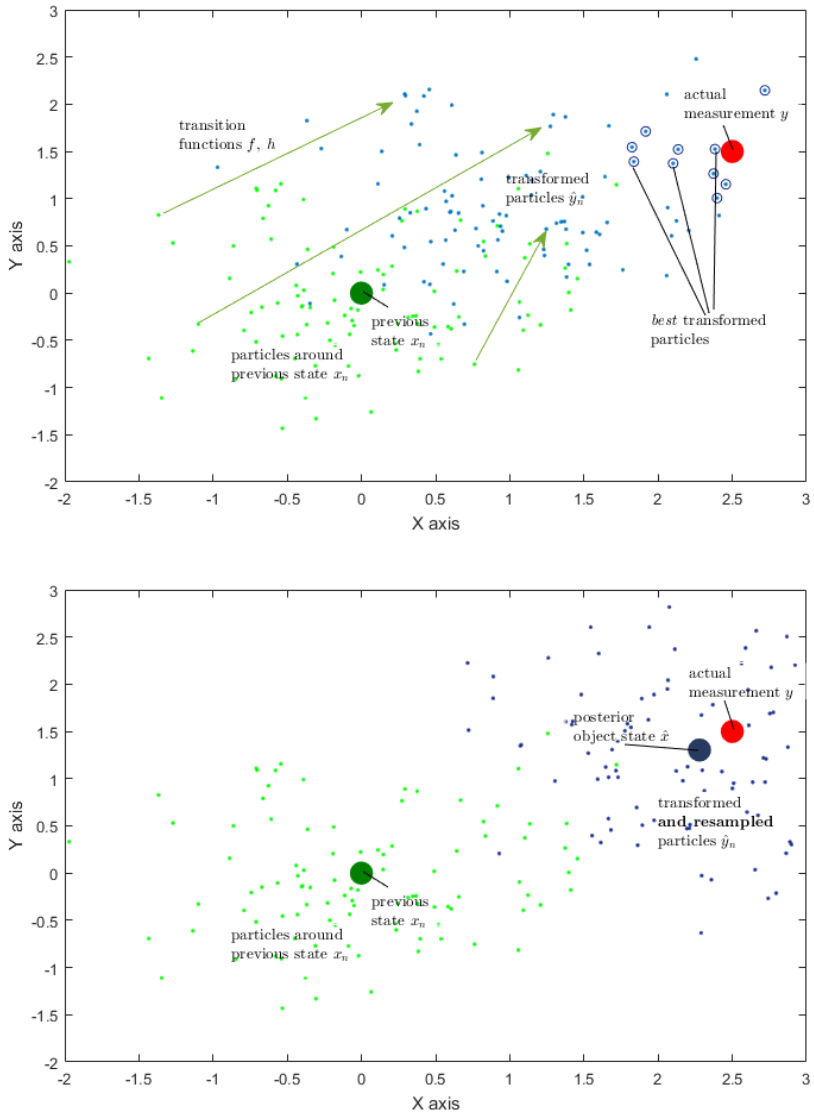

Figure 9: *Schema for the function of the particle filter with 100 particles, $x, y \in Y$*
Top: *Creation of particles around the previous object state and their transformation.*
Bottom: *Resampling of the particles with respect to their weighting based on the actual measurement and estimation of the posterior object state.*

4.4 Limitations

In order to be able to compare the performance of the pure algorithms, we decided to implement the basic filter algorithms only and renounced adding heavy improvements. For many of the limitations below, solutions are available. These will be pointed out in later chapters.

4.4.1 Transition Functions

The accuracy of the filters estimates hardly depends on the accuracy of the transition functions (object state and measurement) used. **Figure 10** schematically illustrates the advantage of a more precise transition function f over a less precise one: The error covariance R can be chosen lower and therefore the estimates will be more accurate. This applies for both, model transition function and measurement transition function.

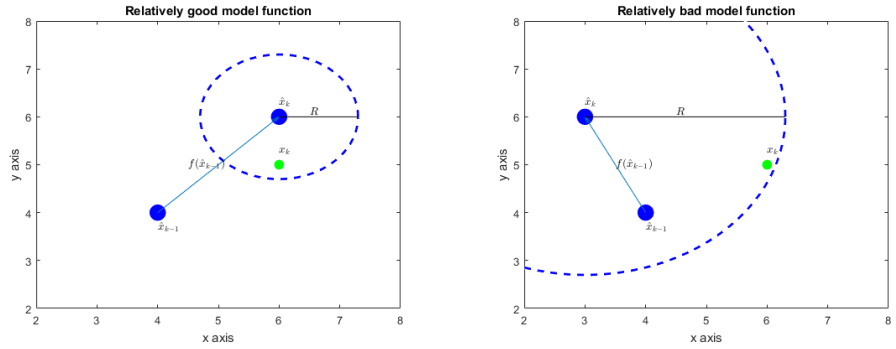

Figure 10: Left: *Relatively good model function f. Noise covariance R can be low.*
Right: *Relatively bad model function f. R must be higher.*

4.4.2 Agile vs. Passive Object Movement

The higher the speed and/or the agility of the object, the better the transition functions have to be in order to ensure the estimations quality. Otherwise, the error covariance matrices must be increased with the same effect as described in **chapter 4.4.1**.

4.4.3 High Noise Level

A high noise level (= low SNR) leads to wrong measurements in the meaning of e. g. *incorrect positions* or *too large or too low TDOA* values.
Figure 11 illustrates the difference between an audio record without any noise and a static SNR of $5dB$. Of course this applies not to audio signals only but signals in general, too. The noise level is not necessarily constant, but could change over time, e. g. different noise volumes of a projector's fan.

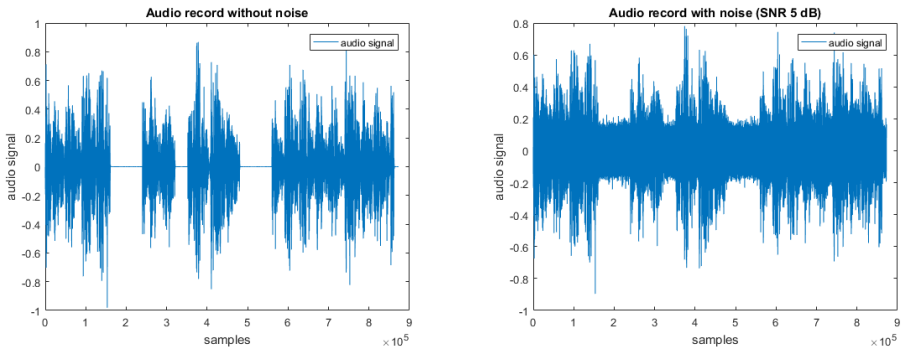

Figure 11: Left: *Audio record with no noise.* ***Right:*** *The same audio record with noise added (SNR* $5dB$*).*

4.4.4 Periods Of Silence

If the object stops producing measurements (e. g. stops speaking) but continues moving, it appears to the filter that there are missing or wrong measurements. There are several approaches conceivable in order to fix or soften this effect:

- Take the measurements as they are (object appearing at another position, **chapter 4.4.5**).
- Stay static at the last known position.
- Use the object transition model alone to *simulate* the object's movement.

4.4.5 Rapidly Moving Object

Sometimes it seems that the object seems to move infinitely fast to another position in the space. Often, this is because of missing measurements or *periods of silence* (**chapter 4.4.4**). In *multi speaker* scenarios, another reason could be the mislabeling of two or more objects.

5 Experiments

In order to check the performance of each individual filter, we performed a series of experiments with different scenarios. We decided to use two different approaches: On top of **SRP pre-localized** directions and a pure **TDOA approach**. This chapter details the used bases for the experiments and justifies the parameters we chose during the experiments.

5.1 Setup

5.1.1 Signal Response Power, Pre-Localized

In order to check the filters' tracking only performances we decided to pre-localize the speaker using *SRP-PHAT* (**chapter 1.4**). We used the SRP implementation of Lathoud [5] which he -and several other researchers- also uses in his AV16.3 corpus [6]. As mentioned, SRP is a grid based algorithm. We defined a hexahedron as grid matching our defined scenario room with a resolution of $0.1m$. This means that each of the grid points has a distance of $0.1m$ to each of its neighbors. *Neighbor* in this case means the direct neighbors on the x, y and z axis excluding diagonals.

Even if SRP estimates the object's position in Cartesian coordinates, we experienced that these are not reliable enough. This is because the algorithm tries to maximize the SRP value over all available grind points. If one has a relatively rough grid (as we have with a resolution of $0.1m$) it is possible, and quite likely, that the algorithm finds maximized grid points differing from the Cartesian ground truth. Having a closer look to **equation 6** this effect is easy to guess, since the SRP value only depends on the recorded signal ($x_i(t)$, $x_j(t)$) and the TDOA between the two microphones ($\tau^d_{i,j}$), but not directly on the position. Even the TDOA does not depend directly on the positions, but on the distance to the microphones (**equation 2**) or the direction of arrival (**DOA**, **equation 3**). The direction of arrival (azimuth of spherical coordinates) is a good and reliable value to measure and therefore we used it as basic unit for this experiment and the following experiments.
Figure 12 shows the difference in accuracy between the Cartesian coordinates and the DOA (azimuth only) of the SRP estimates of a static speaker.
Of course, one could increase the resolution of the SRP estimation, but this leads to an increasing time of execution. As an example: A room of size $5m \times 3m \times 2.5m$ and a resolution of $0.3m$ leads to nearly $1,700$ grid points. The same room with a resolution of $0.2m$ leads to about $5,400$ grid points. We experienced that there is no big difference in the result between the resolutions $0.3m$ and $0.2m$. A good resolution of about $0.01m$ produces about $37,000,000$ grid points. Each of these grid points has to be checked for maximum in each step, i. e. for every audio frame. We experienced that the resolution of $0.1m$ tends to be a good compromise between execution time and accuracy.

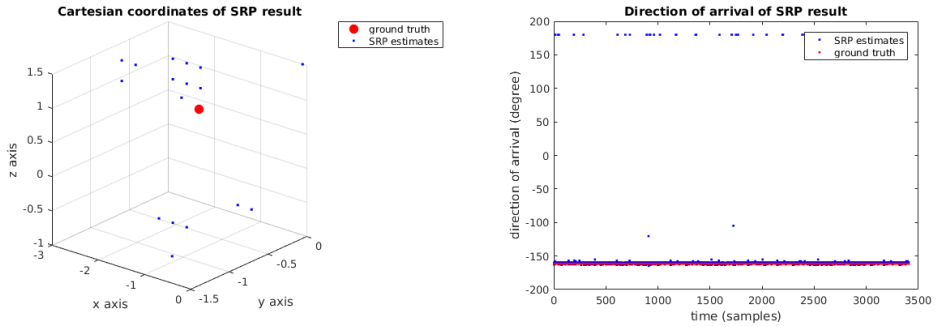

Figure 12: *SRP estimation vs. ground truth of a* ***static*** *speaker.* ***Left:*** *Cartesian speaker estimation* ***Right:*** *Direction of arrival to speaker*

5.1.2 TDOA Approach

In contrast to the SRP based approach we also decided to use a more direct TDOA approach. Following this approach, we *pruned the localizing step* before the tracking and more or less move it to the filter part. It is important to recognize that each filter does *not localize the object directly*, but compares and qualifies the actual measurement with the measurement predicted by its transition function and its predicted speaker state. Therefore, a grid, as used in the SRP approach, is no longer required.

Besides the obvious advantages (execution complexity) of this approach we hoped to achieve *better results in more difficult scenarios*, e. g. scenarios with periods of silence, because of the tracking function of the filter and not only localizing the speaker.

5.1.3 Transition Functions

Since we wanted to keep the experiments as generic as possible, we decided to renounced on accurate object state transition functions. Of course it would be better, in the sense of error rates, to use a transition function that is actually closer to the speakers movement, but in general it is hard, if not impossible, to predict a speakers movement before the scenario starts.

This is why we decided to take a static object model, i. e. a static object transition function:

$$f_{SRP}(x_k) = f_{TDOA}(x_k) = x_k \tag{31}$$

Since the filtering on top of SRP is a *direct measurement* we therefore also used a static transition function:

$$h_{SRP}(\hat{x}_k) = \hat{x}_k \tag{32}$$

For the *indirect measurement* (TDOA) approach we used the TDOA formula as

described in **equation 3**:

$$h(\hat{x}_k) = \tau_{m_i,m_j}(\hat{x}_k) \tag{33}$$

where $\hat{x}_k = \begin{bmatrix} \Theta \\ \Phi \end{bmatrix}$ is the object state to track, Θ describes the direction of arrival (azimuth) and Φ the elevation of arrival.

Annotation: Although we are mainly interested in the azimuth, we also track the elevation. This is needed because the time delay formula depends on both, the azimuth *and* the elevation.

5.1.4 Parameters

Of course, the used parameters had a high impact on the result of each of the scenarios. The most important ones for KF, EKF and UKF are object state covariance P and object state and measurement transition noise covariances Q and R. The Particle Filter does not have a state covariance, since it does not assume Gaussian state distributions. Besides Q and R, here, the important parameters are the number of used particles n and their initial variance v.

As described in **chapter 4.2.6**, Q and R could get calculated. Because we used static models, although the speaker is not static in most of the scenarios, the calculated matrices are not quite valid. They were good clues, but in general we estimated them experimentally based on the calculations done.

$P = I$ established as a solid value for the initial state covariance matrices, where I is the identity matrix. For the **SRP approach**, the following noise covariance matrix intervals established as solid choices, depending on scenario and SNR:

$$\begin{aligned} Q &\in [I \cdot 10^{-3}, ..., I \cdot 10^{0}] \\ R &\in [I \cdot 10^{2}, ..., I \cdot 10^{5}] \end{aligned} \tag{34}$$

For the **TDOA approach**, the following matrices produced valid results.

$$\begin{aligned} Q &\in [I \cdot 10^{-5}, ..., I \cdot 10^{-1}] \\ R &\in [I \cdot 10^{-7}, ..., I \cdot 10^{-4}] \end{aligned} \tag{35}$$

Bavdekar et al. describe in [16] a way on how to estimate Q and R using again a separate EKF.

For the Particle filter, an initial variance of $v = 2^\circ$ and the usage of $n = 100...500$ particles were sufficient and produced reasonable results.

5.2 Experiments

It follows a selection of results from performed experiments. Before running each of them, we made some expectations:

- Since the *transition functions* in the SRP based scenarios are *static* and therefore implicitly *linear*, the EKF reduces to a normal KF (**chapter 4.3.2**).
- The results of the *UKF should be relatively close to the EKF*, since it is an alternative based on the same assumptions.
- The *PF should perform slightly worse* compared to the other two filters, since we used relatively few particles.

Also, we considered a Rooted Mean Squared Error (**RMSE**) of 20° as too uncertain and therefore as *failed*.

5.2.1 Static Scenario

The most interesting part in this experiment is, that without a small amount of noise, SRP-only fails in terms of a too big RMSE, even without periods of silence.

		static	static with 1 period of silence	static with 3 periods of silence
No noise	SRP only	22.7205	21.6001	22.0277
	SRP KF	0.1043	0.0961	0.0969
	SRP EKF	0.1043	0.0961	0.0969
	SRP UKF	0.1043	0.0960	0.0968
	SRP PF	4.1607	3.8306	4.5292
	TDOA EKF	3.4961	7.0544	13.4027
	TDOA UKF	3.7068	4.2448	13.9929
	TDOA PF	8.2679	16.8695	56.8644
SRP $30dB$	SNR only	10.7616	34.3544	50.6412
	SRP KF	0.0697	0.2296	0.6492
	SRP EKF	0.0697	0.2296	0.6492
	SRP UKF	0.0696	0.2293	0.6488
	SRP PF	4.4355	15.4373	42.7201
	TDOA EKF	7.2177	6.9277	10.3640
	TDOA UKF	7.3532	4.9138	8.1450
	TDOA PF	19.2738	75.7083	86.3967

***Table 2: RMSE** ($^\circ$) of static scenario **without noise** and with **small amount of noise** (SNR 30).*

As **table 2** shows, a small amount of noise helps SRP to perform better. This is because of missing/failed measurements in the no-noise part which causes the algorithm to use its *fall-back* plan, i. e. use 0° as estimated location. This also explains the relatively constant RMSE when looking additionally at the two audios with periods of silence.

While adding a small amount of noise (SNR $30dB$) helps the SRP-only algorithm in the case of the scenarios without silence periods, it is not useful for the latter two.

Each of the Kalman Filters (KF, EKF, UKF) performed extremely well, in fact and as expected, KF and the EKF produced the exact same result on top of SRP and the UKF is extremely close to them. This is unique in this thesis because it is the only case where the used model is close to the scenario: *Static speaker* and *static transition functions* f and h. That is why the filters could trust the model more and the measurement less than in the other scenarios, resulting in these good results.
The deficit of the Particle Filter is bigger than expected, especially for the case with low noise.

Obviously, the TDOA approach in general is worse than the SRP approach, especially in the more difficult scenarios with silence periods.
For the other noise levels the error rates were analog to the examples provided, increasing with the noise and periods of silence. Therefore we renounced to report them at this point.
Figure 13 exemplary shows the three experiments done. Firstly the SRP-only, followed by the filtered one on top of the SRP-only results and finally the SRP independent TDOA approach. Note that for the azimuth Θ it does apply that $180^\circ = -180^\circ$. Therefore the visual peaks in the EKF-TDOA approach of **figure 13** are only that high because of the plotting range.

5.2.2 Non Linear Scenario

Because of the relatively low level of nonlinearity, this scenario turned out to be relatively easy to handle for the filters. Again, KF and EKF produced the same result (SRP-based) and UKF is pretty close to this result, too.
As **table 3** clearly shows, putting a filter on top of SRP is beneficial in every case. Looking at the normal records only (without the periods of silence), the RMSE of the SRP-only experiments are approximately 6 times higher and more. The experiments **without noise** produced good to excellent results for both approaches, except for the PF with silence periods.
Even with a **low SNR** of $5dB$ the SRP-based results are acceptable, while the direct TDOA approach already fails at a decent level of $15dB$.

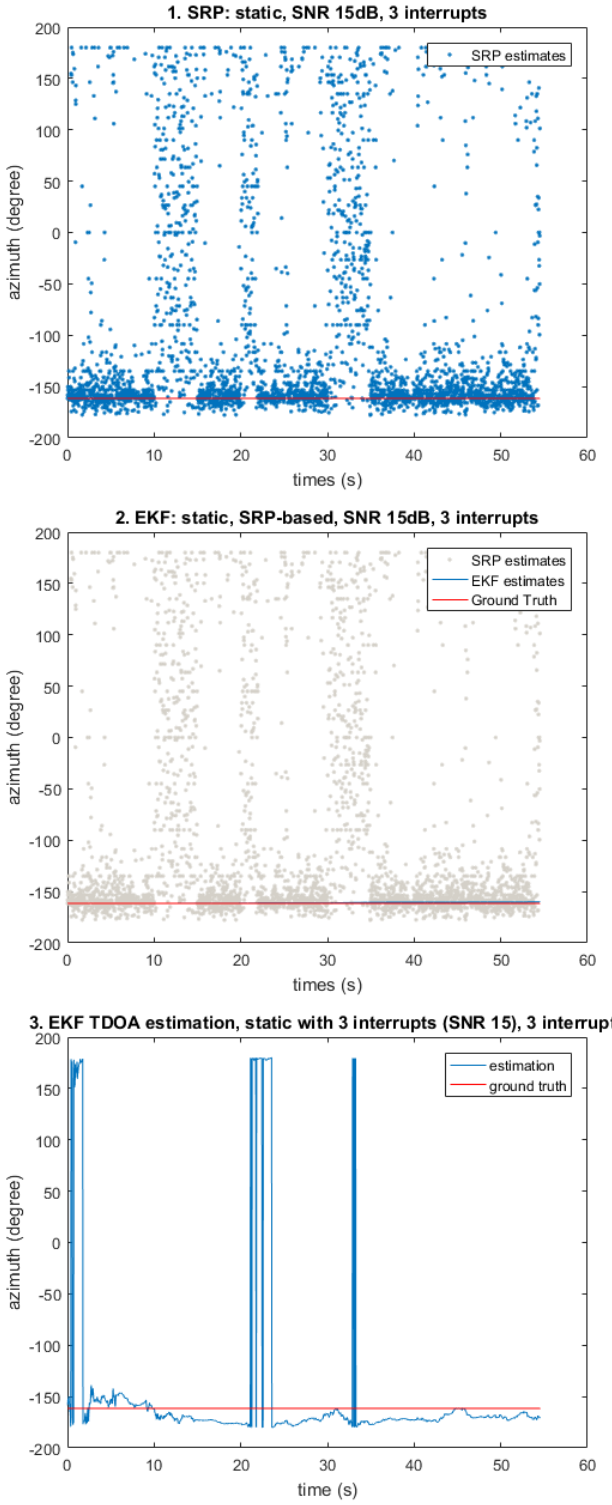

Figure 13: *Static scenario with decent noise (SNR* $15dB$*).*
1. *SRP only.* ***2.*** *EKF on top of SRP.* ***3.*** *EKF with TDOA approach.*

		static	static with 1 period of silence	static with 3 periods of silence
No noise	SRP only	19.9634	20.4864	19.6605
	SRP (E)KF	2.0008	2.3575	2.5084
	SRP UKF	1.9995	2.3565	2.5074
	SRP PF	2.3350	2.4196	2.6982
	TDOA EKF	2.8319	12.1138	11.4902
	TDOA UKF	2.7936	9.5803	11.5289
	TDOA PF	4.2205	57.6764	84.3495
SNR $30dB$	SRP only	8.3895	36.0819	50.7440
	SRP (E)KF	1.9947	13.9869	12.7660
	SRP UKF	1.9948	13.9859	12.7648
	SRP PF	2.6584	21.7404	16.0266
	TDOA EKF	11.0538	20.9717	25.1544
	TDOA UKF	10.8936	17.3914	28.4501
	TDOA PF	14.9138	60.4819	77.6691
SNR $15dB$	SRP only	30.4364	42.7577	54.5943
	SRP (E)KF	5.5962	14.5356	13.8382
	SRP UKF	5.5908	14.5320	13.8340
	SRP PF	9.8469	17.9281	10.7257
	TDOA EKF	23.6184	40.7160	50.7616
	TDOA UKF	35.6637	41.8068	95.9002
	TDOA PF	80.2830	81.8513	77.4901
SNR $05dB$	SRP only	58.8280	61.6458	67.2944
	SRP (E)KF	11.7107	18.4976	14.1200
	SRP UKF	11.6869	18.4988	14.1208
	SRP PF	22.2744	32.5907	33.9102
	TDOA EKF	36.4392	38.6488	54.9348
	TDOA UKF	85.2814	98.3634	84.8097
	TDOA PF	74.4079	80.3600	77.2603

***Table 3: RMSE** (°) of nonlinear scenario **without noise** and with **different levels of noise** (SNR $30dB$, $15dB$, $5dB$).*

5.2.3 Circle Scenario

Table 4 shows the results of the most interesting parts of the experiments performed. For the lower SNRs we renounced to display the results of the TDOA-approach filters, since we considered them as failed.
Again without any noise, the SRP-only results are relatively constant, while as soon

		static	static with 1 period of silence	static with 3 periods of silence
No noise	SRP only	12.83	11.91	12.62
	SRP EKF	10.02	9.49	9.99
	SRP UKF	10.02	9.49	9.99
	SRP PF	19.95	19.8	19.71
	TDOA EKF	9.78	18.5	24.29
	TDOA UKF	12.19	17.86	37.36
	TDOA PF	11.32	78.47	72.59
SNR $30dB$	SRP only	7.89	34.97	52.39
	SRP EKF	9.81	21.34	36.32
	SRP UKF	9.81	21.34	36.32
	SNR PF	20.59	34.26	57.35
	TDOA EKF	13	34.53	40.48
	TDOA UKF	24.13	24.95	27.91
	TDOA PF	17.2	63.99	77.24
SNR $15dB$	SRP only	27.69	41.75	55.11
	SRP EKF	11.23	13.4	33.4
	SRP UKF	11.23	13.4	33.4
	SRP PF	26.15	37.77	58.58
SNR $05db$	SNR only	54.68	60.77	68.32
	SRP EKF	15.81	17.37	25.82
	SRP UKF	15.81	17.38	25.82
	SRP PF	47.72	52.02	70.62

***Table 4: RMSE** (°) of circular scenario **without noise** and with **different levels of noise** (SNR $30dB$, $15dB$, $5dB$).*

as we added some noise to it, the error gets bigger the more periods of silence are in the signal. And again, as in the other experiments, EKF and UKF are close to each other, except for some exceptions, i. e.the TDOA approach with SNR $30dB$.

Figure 14 shows the difference between a SRP based UKF with heavy noise (SNR $5dB$) and a TDOA based PF without any kind of noise. In this case, the overall RMSE of the non noised scenario with PF was slightly better than the overall RMSE of the heavy noised scenario with the UKF, which clearly shows the dominance of the SRP based algorithms.

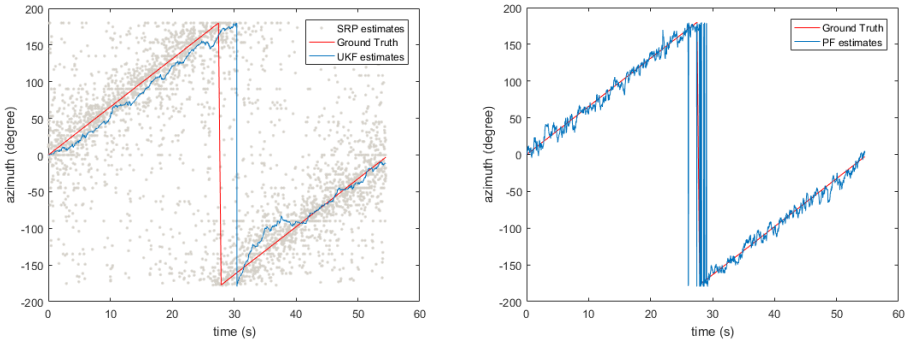

Figure 14: *Circular moving speaker with high noised (SNR* $05db$*,* ***left****) SRP approach and TDOA-based PF without noise (****right****).*

5.2.4 Linear Scenario With Sudden Appearance

This experiment was designed to be the hardest on purpose. The main difficulty was the speaker's rapid movement through the room and sudden appearance at another position. Therefore the velocity of the speaker is not static anymore and causes trouble, especially for the filters' tracking function. Goal of this experiment is to check the filters' agility in hard scenarios like that.

		static	static with 1 period of silence	static with 3 periods of silence
No noise	SRP only	7.6248	8.5693	10.1954
	SRP EKF	12.5693	12.2114	12.5148
	SRP UKF	12.5699	12.2120	12.5154
	SRP PF	24.7676	25.3689	24.6815
	TDOA EKF	97.8132	95.6738	91.8924
	TDOA UKF	54.0591	54.2657	57.8579
	TDOA PF	84.9758	90.8512	83.9620
SNR $30dB$	SRP only	6.1573	34.8808	66.0711
	SRP EKF	14.3757	35.1916	46.3026
	SRP UKF	14.3762	35.1918	46.3028
	SNR PF	27.1731	32.5813	62.0624
	TDOA EKF	69.5716	90.6113	54.2002
	TDOA UKF	81.7156	89.1739	88.5572
	TDOA PF	79.9628	81.8218	77.4885

Table 5: RMSE *(°) of linear moving scenario with sudden appearance at another position* ***without noise*** *and a low noise of SNR* $30dB$*.*

Table 5 shows that this scenario was in fact really hard, especially for the TDOA

approach filters.

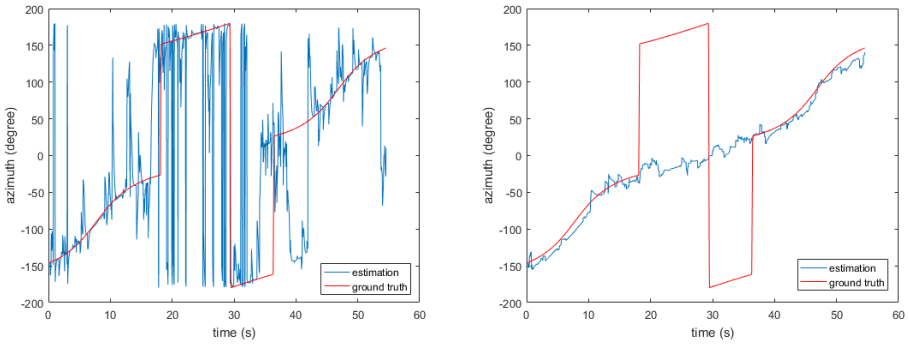

Figure 15: *Linear moving speaker with jumps at a low noise level* ($30dB$)*.* ***Left:*** *Minimal overall RMSE, EKF.* ***Right:*** *In places minimal RMSE, EKF.*

Figure 15 illustrates *why* this scenario is so hard to handle. While both graphs show the exact same scenario (SNR $30dB$, TDOA-based approach, EKF), they differ in the used parameters. The left one is adjusted to produce a minimal RMSE over the *entire signal*, while the right one tries to be as near to the ground truth as possible. In other words: The left one *trusted* the measurements more than its model and the right one the other way around. This results in a very agile filter on the left and a relatively passive filter on the right.
Using a *pre-localized SRP-based* version (**figure 16**) provides results that are much better.

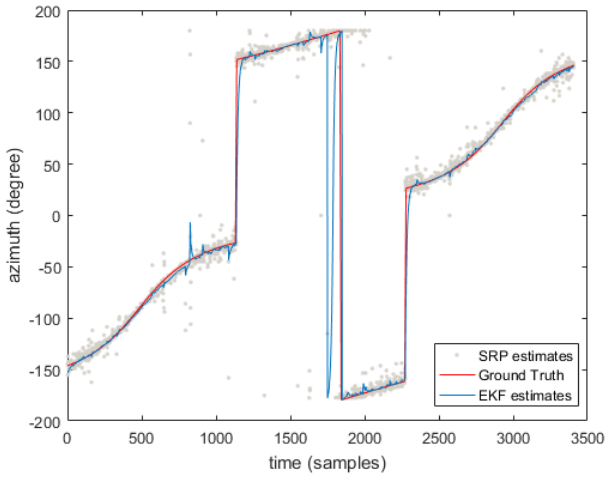

Figure 16: *Linear moving speaker with jumps at a low noise level* ($30dB$)*, SRP-based EKF.*

Again, it is $180° = -180°$ for the azimuth. Interesting are the *round edges* at every jump. This is caused by the relative balanced parameters, meaning the trust in the

model and in the measurement is relatively the same. This leads to a decent overall RMSE rate of 14.38 degree.

5.2.5 Overall Results

In oder to evaluate the filters overall performance we computed the mean RMSE for each approach, filter and scenario. **Table 6** shows the result. As expected before and experienced during the experiments, the Unscented Kalman Filter produced almost similar RMSE values to the Extended Kalman Filter. This can be seen extremely good on the SRP approach, no matter what amount of noise added, and also for the TDOA approach with reasonable amount of noise. Having higher noise levels, the results are drifting more apart, but failed anyways.

	No noise	SNR $30dB$	SNR $15dB$	SNR $5dB$
SRP only	16.25	34.92	42.25	59.86
SRP (E)KF	5.9990	13.3764	13.6189	16.5904
SRP UKF	5.9985	13.3753	13.6169	16.5906
SRP PF	12.1211	24.4567	31.9609	65.5715
TDOA EKF	12.7583	23.0631	52.0023	71.3110
TDOA UKF	13.0934	24.5367	74.0301	90.0963
TDOA PF	65.1319	76.4721	80.4673	76.2049

Table 6: *Mean RMSE ($^\circ$) of all scenarios with respect to the noise levels.*

Looking at SRP-only and pure RMSE values, each of the filter successfully decreased the error in each case, except for the Particle Filter for high noise scenarios. In general, the Particle Filter fails earlier than any of the other filters. The same applies for the more direct TDOA approach. While the Particle Filter failed even without noise, the others brought an improvement compared to SRP only, at least for low noise scenarios. We discussed the cause of this behavior during and after the experiments.

6 Discussion

During and after the experiments, we briefly discussed the implementations and the RMSE results for each scenario.

6.1 Kalman Based Filters

As mentioned in **chapter 5**, the Kalman Filter and the Extended Kalman Filter produce the same result for linear models. This is not only valid for the scenarios we used during the experiments, but a generic rule. This fact can easily get proven by looking at the definitions of the algorithms (**chapter 4.3.1** and **chapter 4.3.2**): If the model functions f and h are linear, then there are matrix representations available, F and H, respectively. This means that there is no linearization needed to get F and H and the Extended Kalman Filter reduces to the normal Kalman Filter.

As seen in the results, the UKF was almost always extremely close to the EKF. In the experiments section we argued that the the UKF is designed as an alternative to the EKF. At this point we want to detail this a bit more. Looking at the algorithm definition (**chapter 4.3.3**) one can see that the filter's frame stays the same as of the EKF. In contrast to the EKF's linearization approach, the UKF computes so called *sigma points* and uses them to compute the predicted mean and the predicted measurement statistics. Roughly, this process is the same as of the EKF, but with different equations since of the usage of sigma points.
This means that the success (or failure) of the UKF depends on the accuracy of the sigma points used. The biggest advantage with respect to the EKF is that there is no need of computing the linearizations of the model functions, i. e. computing *Hessian* and *Jacobian* matrices, which can be hard, even with computer aided tools such as MATLAB. That is why the Unscented Kalman Filter has its raison d'être besides the other filters. In our experiments the Jacobian matrices were relatively easy to compute and there was no reason to compute Hessian matrices, but in different scenarios and different tracking states the UKF can be a good alternative with accuracy close to the EKF [1] while being easier to implement.

6.2 Particle Filter Performance

We expected that the Particle Filter would perform worse than the other filters. But the experiments showed much worse results, especially for the TDOA based scenarios with periods of silence. A reason for this is because of the used multivariate Gaussian distribution to *estimate the particle weights*: If the actual measurement is too far from the particles, the distribution to compute the weights evaluates close to zero for each particle. Normally at this point, the normalization would compensate this problem, but in practice, at some point, the computer's (or MATLAB's) maximal resolution is reached, resulting in the weight 0. In the worst case this is true for each particle, resulting in an undefined weight (division by 0) and a fall back

plan is needed, e. g. take the *measurement* as posterior object state (if possible) or take the last object state as the new one (compare **chapter 4.4.4**). In fact this is why the Particle Filter performs that bad, especially when there are periods of silence and/or moderate-to-heavy noise is included. There are several approaches to eliminate this behavior. Some of them, e. g. the so called *Extended Kalman auxilary Particle Filter for single-object tracking*, are described and discussed in [1]. Another possibility would be to modify the methods calculating the weights and estimating the posterior object state. Cho et al. [12] for example provide an extension of the Particle Filter that recognizes when it is useful to resample the particles. But, remembering that we used a static function as object transition model, in our case a more detailed transition model would be advantageous and lead to better results.

Besides the filter modifications, there are methods to increase the measurement accuracy as well, since often missing or failed measurements could get easily recognized and fixed. Oualil et al. [3] for example provide a Gaussian mixture model for improving TDOA based acoustic source tracking.

6.3 RMSE As Our Measurement Criterion

The framework for the experiment was designed to compute the minimal RMSE for each scenario and filter on a predefined parameter interval. This is why we decided to use the overall RMSE as our main criterion, knowing that this is not meaningful in every single case.

Figure 17 exemplary shows why the overall RMSE is not always the best choice as criterion. While $estimation\ 1$ has a minimal lower overall RMSE of 0.1160 than $estimation\ 2$ with 0.1375, the result of the second one is much *smoother*, even without applying a dedicated smoother, especially for the second half.

Another important criterion is the **recovery time**, which we defined as following: *The time after a period of silence (or of failed measurements) the filter needs until it reaches the overall RMSE again.*

It remains to the future work to extend the framework in considering additional criterion to the overall RMSE.

6.4 Execution Time/Filter Complexity

Execution time in general is hard to define and measure, since it hardly depends on the used computer setup, as well as on the implementation of the algorithms itself. As an example, there are several implementations of a discrete Fourier Transformation, all with different benefits and disadvantages, some of them highly paral-

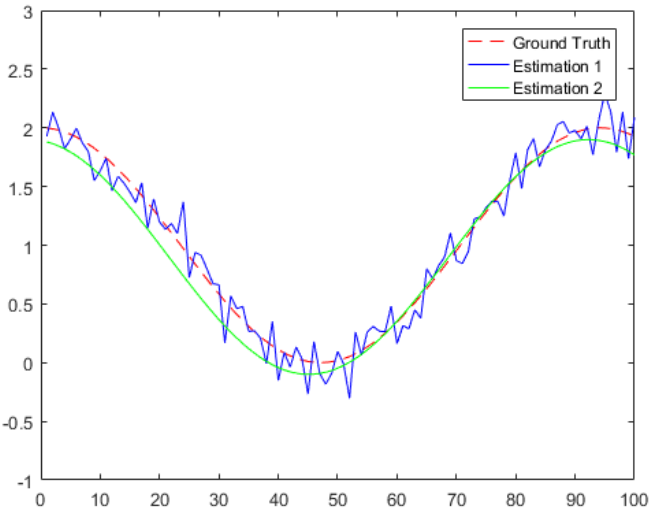

Figure 17: *Overall RMSE (unit-less) of two estimations with respect to the ground truth.* 0.1160 *for estimation 1,* 0.1375 *for estimation 2.*

lelized, others not at all.

But in order to be able to rate the speed of the filters or at least compare their complexities, we decided to estimate their complexity using the Big O notation ($\mathcal{O}$).

Annotation: Since we want to compare the filters themselves, we assumed that matrix operations are linear to the dimensions of the participating matrices. For the DFT we assumed a complexity in $\mathcal{O}(f \cdot \log(f))$ that is well known reached by the FFT (f is the size of the array to transform). For each algorithm we considered a single time iteration.

6.4.1 Signal Response Power

Having M microphones ($\Rightarrow P = \begin{pmatrix} M \\ 2 \end{pmatrix}$ unique microphone pairs) and N possible states in the grid, the complexity of SRP for each time iteration is at least $\mathcal{O}(P \cdot N \cdot 2f \cdot \log f) = \mathcal{O}(P \cdot N \cdot f \cdot \log f)$.

6.4.2 (Extended) Kalman Filter

Having an object state dimension of d_x and a measurement dimension of d_y and additionally assuming that the model functions (and their linearization processes) are constant, the Kalman Filter and its extension is at least in $\mathcal{O}(n)$ with $n \sim d_x + d_y$.

6.4.3 Unscented Kalman Filter

Let there be s sigma points where $n = s \sim d_x + d_y$, the Unscented Transformation is in $\mathcal{O}(s)$. Then the UKF is at least in $\mathcal{O}(2s) + \mathcal{O}(5s) = \mathcal{O}(n)$, putting it to the same complexity class as the standard Kalman Filter and its extension.

6.4.4 Particle Filter

Obviously the complexity of the Particle Filter heavily depends on the number of particles p. Again, we assumed that the model functions are constant. The filter itself contains at least 4 sums over p ($= for-loops$) so that the algorithm is at least in $\mathcal{O}(4 \cdot p)$. Considering additionally the Gaussian distributions, and therefore their array operations that are linear to the object state and measurement dimensions (n so that $n \sim d_x + d_y$), the complexity increases to at least $\mathcal{O}(p + p \cdot n) = \mathcal{O}(p \cdot (1 + n))$.

7 Conclusion And Future Work

Finally, we can say that the experiments showed that the filters on top of SRP are highly beneficial in most of the cases. A more direct TDOA approach did not perform as good as expected. While the results for scenarios with low noise and without interrupts are okay, the other scenarios failed fast. Reasons for these failures are predominantly missing or wrong measurements, as well as periods of silence. Of course, the parameters (covariance matrices etc.) are important, too, as well as the model functions, but we came to the conclusion that it is important to process the measurements itself. A possible approach for this could be eliminating measurements that are clearly wrong. We observed, proportional to the noise level, that the SDOA measurements are sometimes totally wrong in the term that *GCC* provides a delay of more samples than physically are possible. These -obviously wrong- measurements could get eliminated easily, e. g. using a *Low Pass Filter*. Another approach uses the filter's knowledge of the estimation and the measurements in the past, e. g. using a Gaussian mixture model as proposed by Oualil et al. [3].

The implementation of each filter in general was mostly straight forward. The only difficulties occured when adapting the filters to our models, meaning the model functions and their processing (linearization etc.) as well as the handling of the object state and measurement dimensions.

Looking at the overall RMSE only, a combination of SRP and a (Extended/Unscented) Kalman Filter on top provides the best results. While the computational effort for this solution is enormous, each direct approach via TDOA is much faster (in terms of complexity), but provides worse results. Indeed we did not perform any computational time study, but we experienced that the approaches on top of SRP took up to 15 times the time than the TDOA approaches. Knowing this, one can choose between these two approaches, depending on what is needed in a certain scenario.

As the experiments and their evaluations show, each of the filters is able to produce reasonable results when respecting some parameters as noise level and speaker movement. But in praxis one hardly knows the scenario and its frame parameters before it happens. In general we can say that, if execution time is not critical, i. e. if offline tracking is sufficient, the SRP-based filters are the best choise. In situations, where online tracking is needed, the TDOA-based filters (including diverse optimizations) provide tracking performance that is obviously worse than the SRP-based ones, but in much less time.

Besides the overall RMSE, there are several other criterions usable to measure the quality of the result. By the help of the recovery time, for example, one can measure how *agile* the filter is, but a unit for how *smooth* an estimation is, could also be

meaningful in some scenarios. A possibility for this would be to iterative calculate the (mean) elevation of an estimation. The more homogeneous this elevation (in combination with the RMSE) the smoother the estimation.

This thesis considers only with single-object tracking filters, ignoring multiple-object ones. As mentioned in the seminar paper [7] there are in general two possible approaches to achieve multi-object support:
Firstly, there exists the dedicated distinction followed by applying the single-object filters, one for each object. And on the other side, one could use a multi-object filter such as the PHD filter. The framework built in this thesis is basically capable of running multi-object scenarios. It remains to the future work to implement multi-object scenarios and execute appropriate filters. Of course, any other filter, or extensions to the existing ones, could get tested with the thesis' framework as well.

Another constraint we defined before running the experiments was to consider the azimuth only. Currently the framework is able to track the azimuth as well as the elevation to the object, but the RMSE optimization only considers the azimuth. A future step would be to consider the elevation as well.

Another scenario to check in the future could be varying the noise: Different SNR levels in the same scenario with static (or dynamic) noise covariances, for example, mime more realistic circumstances such as a projector fan in the room.
Besides the different levels of noise one could run experiments with non-Gaussian noises. In this case one has to make sure that the used filters are capable of handling non-Gaussian noise.
And last but not least one could vary the audio files used as basis for the scenarios. As mentioned, we used a audio file where the speaker talks without any period of silence, so that we are able to define periods of silence when and especially where we wanted. Using other audio files with more natural talking speakers remains to the future work.

List of Figures

List of Tables

List of Algorithms

Appendices

A Filter algorithms

This section provides our implementations of the filters used during this thesis. Please be informed that the filters -as shown here- only consist of the *prediction* and *update* step for a single measurement (direct measurement or an audio frame). In order to keep it as simple as possible we pruned all non filter related code (e. g. code to plot) in this document. Actually there is a wrapper around each of the filters that handles the measurement acquisition, estimation storage, error calculation and other important things. Generally spoken, the general filtering process is described in **algorithm 6**.

```
1 Initialize the filter's individual parameters
2 For each iteration (e. g. each incoming measurement)
3     Call the filter
4         Predict the object's state on base of the last prediction.
5         Predict the object's next measurement on base of the predicted state.
6         Correct the prediction using the incoming measurement.
7     End call
8     Handle returns and store estimation.
9 End for each
```

Algorithm 6: Generic filtering process in pseudo code.

A.1 Kalman Filter

Subhash Challa [1]

```
function [x,P] = Kalman_Filter (x,y,F,H,P,Q,R)
  %% KALMAN FILTER
  % Inputs:
  % x: The last predicted state
  % y: The object's measuremetns (what the sensor measured)
  % F and H: transition matrices
  % P: state covariance matrix
  % Q and R: error covariances matrices (model and observation)
  % Outputs:
  % x: State estimate
  % P: Posterior state covariance matrix at time k+1

  % -- Step 1: predicted mean (x_hat_k),
  %            covariance matrix (P_k)
   x_hat_k = F * x_k_1;
   P_k = F * P * F' + Q;

  % -- Step 2: predicted measurement,
  %            innovation covariance matrix (S_k),
  %            Kalman gain (K_k)
   y_hat_k = H * x_hat_k;
   S_k = H * P_k * H' + R;
   K_k = P_k * H' / S_k;

  % -- Step 3: posterior mean (the "new" x_hat),
  %            covariance matrix (the "new" P)
   x = x_hat_k + K_k * (y - y_hat_k);
   P = P_k_k_1 - K_k * H * P_k_k_1;
end
```

A.2 Extended Kalman Filter

Subhash Challa [1]

```
function [x,P] = Extended_Kalman_Filter(x,y,f,F,h,H,P,Q,R)
  % EXTENDED KALMAN FILTER
  % Inputs:
  % x: The last predicted state
  % y: The object's measurements (what the sensors measured)
  % f and h: transition functions
  % F and H: linearized transition functions (matrix form)
  % P: state covariance matrix at time k
  % Q and R: error covariances matrices (model and observation)
  % Outputs:
  % x: State estimate
  % P: Posterior state covariance matrix at time k+1

  % -- Step 1: Compute the linearized models F (of f) and H (of h)
  % Already done before the filter call (Initialization step)
  % using jacobian(...)

  % -- Step 2: Compute predicted mean and covariance matrix
   x_hat = f(x);
   P_k_k_1 = F * P * F' + Q;

  % -- Step 4: Compute the predicted measurement, innovation
      covariance
  % matrix and Kalman gain
  % IMPORTANT: Also here: Took the normal KFs equation
   y_hat = h(x_hat);
   S_k = H * P_k_k_1 * H' + R;
   K_k = P_k_k_1 * H' / S_k;

  % -- Step 5; Compute the posterior mean and covariance matrix
   x = x_hat + K_k * (y - y_k);
   P = P_k_k_1 - K_k * H * P_k_k_1;

end
```

A.3 Unscented Kalman Filter

Subhash Challa [1]

```
function [x,P] = Unscented_Kalman_Filter(x,y,f,h,P,Q,R)
  % UNSCENTED_KALMAN_FILTER      %
  % Inputs:
  % x: Previous model state
  %   y: Current measurement
  %   f: The model transition function (predict object state)
  %   h: The measurement transition function (predict measurement)
  %   P: Initial value of the estimation P
  %   Q,R: The model/measurement noise covariance
  % Outputs:
  % x: State estimate
  % P: Posterior state covariance matrix at time k+1
  dim_y = numel(y);
  dim_x = numel(x);

  % Static parameters for the filter
  alpha = 1 * 10^-3;                              % atomar, tunable
  beta = 2;                                        % atomar, tunable
  ki = 0;                                          % atomar, tunable
  lambda = alpha^2 * (dim_x + ki) - dim_x;        % scaling factor
  c = dim_x + lambda;                              % scaling factor

  %% 1. Determine sigma points X_k-1^1 ... X_k-1^s and its
  % weights w^1 ... w^s
  W_mean = [lambda / c  0.5 / c + zeros(1, 2 * dim_x)];
  W_cov = W_mean;
  W_cov(1) = W_mean(1) + (1- alpha^2 + beta);
  c = sqrt(c);
  X = calculate_sigmas(x, P, c);

  %% 2. Compute the transformed sigma points (X) +
  %% 3. Compute the predicted state statistics
  [x_1,X_1,P_1,X_2] = unscented_transformation(f,X,W_mean,W_cov,
      dim_x,Q);

  %% 4. Determine sigma points X_k^1 ... X_k^s and its
  % weights w^1 ... w^s to match a mean x_hat_k_k-1
  % and covariance matrix P_k_k-1 +
  %% 5. Compute the transformed sigma points (Y) +
  %% 6. Compute the predicted measurement statistics
  [y_1,Y_1,P_2,Y_2] = unscented_transformation(h, X_1,W_mean,W_cov
      ,dim_y,R);

  %% 7. Compute the posterior mean and covariance matrix
  P_12 = X_2 * diag(W_cov) * Y_2';
  K = P_12 / P_2;
  x = x_1 + K * (y - y_1);
  P = P_1 - K * P_12';
end
```

Yi Cao [11]

```
function [y,Y,P,Y_1] = unscented_transformation(f,X,W_mean,W_cov,n
    ,R)
  % UNSENTED_TRANSFORMATION Calculates the unscented
      transformation
  %   f        nonlinear map (transition function)
  %   X        sigma points
  %   W_mean   weights of mean
  %   W_cov    weights of covariance
  %   n        number of outputs of f
  %   R        additive noise covariance
  %
  %   y        transformed mean
  %   Y        transformed sampling points
  %   P        transformed covariance
  %   Y_1      transformed deviations

  L = size(X, 2);
  y = zeros(n, 1);
  Y = zeros(n, L);

  for k = 1:L
  Y(:, k) = f(X(:, k));
  y = y + W_mean(k) * Y(:, k);
  end

  Y_1 = Y - y(:, ones(1, L));
  P = Y_1 * diag(W_cov) * Y_1' + R;
end
```

Yi Cao [11]

```
function [X] = calculate_sigmas(x,P,c)
  % CALCULATE_SIGMAS Calculates the sigma points around a
      reference point x
  % x      Reference point
  % P      Covariance
  % c      Coefficient
  % X      The calculated sigma points

  A = c * chol(P)';
  Y = x(:, ones(1, numel(x)));
  X = [x Y+A Y-A];
end
```

A.4 Particle Filter

Subhash Challa [1]

```
function [x, particles] = Particle_Filter(x,y,f,h,Q,R,n,v,
    particles)
  % PARTICLE_FILTER
  % Inputs:
  %   x: Previous object state
  %   y: Current measurement
  %   f: The model transition function (predict object state)
  %   h: The measurement transition function (predict measurement)
  %   Q,R: The model/measurement noise covariance
  %   n: Number of particles to use
  %   v: Variance of particles
  %   particles: Previous particles
  % Outputs:
  %   x: State estimate
  %   particles: Posteriour particles

  dim_x = numel(x);
  dim_y = numel(y);

  % generate particles around the initial value of x
  if numel(particles) == 0
    % Init particles for first time
    x_particles = zeros(n, dim_x);
    for i=1:n
      x_particles(i,:) = x + sqrt(v) * randn(dim_x,1);
    end
  else
    % use the last particles
    x_particles = particles;
  end

  % Do the transition for every particle and measurement
  x_particles_hat = zeros(n, dim_x);
  y_particles_hat = zeros(n, dim_y);
  particle_weights = zeros(n, 1);
  for i=1:n
    % Predict the state particles and measurement particles
    x_particles_hat(i,:) = f(x_particles(i,:)) + (sqrt(Q) * randn(
        dim_x, 1))';
    y_particles_hat(i,:) = h(x_particles_hat(i,:));

    % Multivariate normal distribution
    % Without "1/(sqrt(2*pi)^2 * det(R))" since it gets weighted
        anyways
    particle_weights(i) = exp(-1/2 * (y' - y_particles_hat(i,:)')'
        * inv(R) * (y' - y_particles_hat(i,:)'));
  end

  pw_sum = sum(particle_weights, 1);
  for i=1:1:n
```

```
47      particle_weights(i,:) = particle_weights(i,:) ./ pw_sum;
48    end
49
50    % Resample the particles
51    for i=1:n
52      x_particles(i,:) = x_particles_hat(find(rand <= cumsum(
            particle_weights),1),:) + sqrt(v) * randn(1,dim_x);
53    end
54
55
56    % Calculate the mean of the particles which is our final
          estimate
57    x = mean(x_particles);
58  end
```

B Thesis Framework And Tools

This chapter deals with the frameworks and scripts we developed during the thesis and its preparation, worth mentioned. During the entire process we were interested to keep the tools as general in possible. In the ending we now have a framework which allows to define new scenarios and filters with relatively low effort in order to check their performances.

B.1 Simulation Process

As mentioned in **chapter 3.1** we used *fastISM* to simulate our audio files. We wrote a wrapper which allows to define the speaker's trajectory (as *fastISM* audio file) and produces the simulated audio files with different noise levels.

B.2 Experiment Setup

Using the framework, it is possible to run the same filter with a set of different settings over several audio files and/or pre-localized directions. In general, to run the experiments with a custom filter it is needed to define the filter and to define how to use the filter. The remaining parts (scenario definition, audio handling, etc.) is done by the framework.

B.3 Git Repository

We heavily used the version control system Git during the entire thesis and its preparations. A copy of the repository can be requested at the author.

References

[1] Subhash Challa, Mark R. Morelande, Darko Musicki, Robin J. Evans - *Fundamentals of Object Tracking*, November 2011

[2] Ulrich Klee, Tobias Gehring, John McDonough *Kalman Filters for Time Delay of Arrival Based Source Localization*

[3] Youssef Oualil, Friedrich Faubel, Mathew Magimai Doss, Dietrich Klakow - *A TDOA Gaussian Mixture Model For Improving Acoustic Source Tracking*, 20th European Signal Processing Conference (EUSIPCO 2012), August 2012

[4] J. H. DiBiase - *A high-accuracy, low-latency technique for talker localization in reverberant environments using micro-phone arrays*, Ph.D. thesis, Brown University, 2000.

[5] Guillaume Lathoud - `http://glat.info/`, as of August 2016

[6] AV16.3: an Audio-Visual Corpus for Speaker Localization and Tracking - `http://www.glat.info/ma/av16.3/`, as of August 2016

[7] Dominic Buchheit - *Experimental study of object tracking and its applications - Seminar paper*, July 2016

[8] Eric Lehmann - *http://www.eric-lehmann.com/fast_ISM_code/*, as of September 2016

[9] Eric A. Wan, Rudolph van der Merwe - *The Unscented Kalman Filter for Nonlinear Estimation*

[10] Simon J. Julier- *The Scaled Unscented Transformation*, January 1999

[11] Yi Cao - *https://www.mathworks.com/matlabcentral/fileexchange/18217-learning-the-unscented-kalman-filter*, December 2010.

[12] Jeong A Cho, Hanbyeul Na, Sunwoo Kim, Chunsoo Ahn - *Moving-Target Tracking Based on Particle Filter with TDOA/FDOA Measurements*, ETRI Journal, Volume 34, Number 2, April 2012.

[13] E. Lehmann, A. Johansson - *Diffuse Reverberation Model for Efficient Image-Source Simulation of Room Impulse Responses*, IEEE Transactions on Audio, Speech and Language Processing, vol. 18, no. 6, pp. 1429-1439, August 2010.

[14] Simon J. Julier, Jeffrey K. Uhlmann -
Unscented Filtering and Nonlinear Estimation, Proceedings if the IEEE, Vol. 92, NO. 3, March 2004.

[15] E. Lehmann, A. Johansson -
Prediction of energy decay in room impulse responses simulated with an image-source model, Journal of the Acoustical Society of America, vol. 124(1), pp. 269-277, July 2008.

[16] Vinay A. Bavdekar, Anjali P. Deshpande, Sachin C. Patwardhan -
Identification of process and measurement noise covariance for state and parameter estimation using extended Kalman filter, Journal of Process Control 21, 2011