

Masterarbeit

**Laufzeitbasierte Schallortung mit unbekanntem  
Sender- und Empfängerpositionen**



Albert-Ludwigs-Universität Freiburg  
Institut für Informatik  
Lehrstuhl für Rechnernetze und Telematik  
Prof. Dr. Christian Schindelbauer

November 2009

(Rev. October 2012)

**Johannes Wendeborg**



## **Zusammenfassung:**

*Die Lokalisierung von Signalquellen ist eine wichtige Aufgabe in der mobilen Navigation. Eine Methode zur Ortsbestimmung dieser Signalquellen ist die Multilateration, die Auswertung des Laufzeitunterschieds zwischen Empfängern. Zu dieser Methode existiert bereits eine Vielzahl von Ansätzen, bei denen die Position der Empfänger bekannt sein muss. In gewissen Fällen ist jedoch ein Sensornetzwerk wünschenswert, bei dem die Position der Empfänger nicht vorgegeben ist, etwa bei der Geländeaufklärung, in Katastrophenfällen, oder wenn reguläre Ortungsdienste (GPS) nicht verfügbar sind.*

*In dieser Arbeit wird untersucht, ob eine laufzeitbasierte Lokalisierung der relativen Position von Schallquellen möglich ist, wenn die Position der Mikrofone nicht bekannt ist. Die Beantwortung dieser Frage ist in zwei Stränge geteilt:*

- Im ersten Teil der Arbeit wird ein Algorithmus entwickelt, der die Lokalisierung von Schallquellen für eine beliebige Anzahl bekannter oder unbekannter Mikrofone durchführt. Er basiert auf einer physikalischen Simulation mit federbasierten Constraints.*
- Im zweiten Teil wird eine Software ausgearbeitet, die über die Mikrofone von PCs und Laptops Schallsignale aufnimmt und ihnen präzise Zeitstempel zuordnet. Diese dienen als Eingabe für den Algorithmus des ersten Teils.*

*Ein dritter Teil kombiniert die beiden Stränge und gibt eine konkrete Antwort auf die Frage, inwiefern die Lokalisierung von Schallquellen mit PC-Hardware möglich ist. Desweiteren wird ein Ausblick geliefert, in welche Richtung weitere Arbeiten gehen können, um eine robuste, wechselseitige Lokalisierung zu ermöglichen.*

## ERKLÄRUNG

Hiermit erkläre ich, dass ich diese Abschlussarbeit selbständig verfasst habe, keine anderen als die angegebenen Quellen/Hilfsmittel verwendet habe und alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten Schriften entnommen wurden, als solche kenntlich gemacht habe. Darüber hinaus erkläre ich, dass diese Abschlussarbeit nicht, auch nicht auszugsweise, bereits für eine andere Prüfung angefertigt wurde.

\_\_\_\_\_

Datum

\_\_\_\_\_

Unterschrift

## **Danksagung**

Ich danke Prof. Dr. Christian Schindelbauer vom Lehrstuhl für Rechnernetze und Telematik für die freundliche Betreuung meiner Masterarbeit und für die hilfsbereite Unterstützung beim Organisieren von Ausrüstung. Außerdem danke ich Emanuel Koziolek und Frank Nisch für die vielen Stunden sorgsamem Korrekturlesens. An dieser Stelle möchte ich auch Sascha Frank und Natascha Widder für die Unterstützung bei den Experimenten danken, Andreas Kern für seine einsichtigen Ratschläge und Lucas Taeger für das bereitwillige Zurverfügungstellen von Ausrüstung.



# Inhaltsverzeichnis

<b>1</b>	<b>Einleitung</b>	<b>1</b>
1.1	Motivation . . . . .	1
1.2	Gliederung . . . . .	3
<b>2</b>	<b>Themenverwandte Arbeiten</b>	<b>4</b>
2.1	Sound Source Triangulation Game . . . . .	4
2.2	Robust Sound Source Localization . . . . .	6
2.3	Stat-Bat . . . . .	7
<b>3</b>	<b>Lokalisierung</b>	<b>9</b>
3.1	Assoziationsproblem . . . . .	9
3.2	Freiheitsgrade . . . . .	11
3.3	Qualität der Verteilung im Raum . . . . .	13
3.3.1	Multiple Regression . . . . .	13
3.3.2	Abstand von der Ebene . . . . .	14
3.3.3	Kreuzprodukt . . . . .	15
<b>4</b>	<b>Lampshade-Algorithmus</b>	<b>16</b>
4.1	Exkurs: Newtonsches Iterationsverfahren . . . . .	16
4.2	Geometrische Darstellung . . . . .	18
4.3	Physikalische Simulation . . . . .	22
4.4	Programmiertechnik . . . . .	23
4.5	Vergleich mit Ground-Truth . . . . .	25
4.6	Experiment . . . . .	27
4.6.1	Genauigkeit . . . . .	30
4.6.2	Laufzeit . . . . .	30
4.7	Lokale Minima . . . . .	31
4.8	Zusammenfassung . . . . .	35
<b>5</b>	<b>Timestamping</b>	<b>36</b>
5.1	Glättung von Messreihen . . . . .	37
5.1.1	Arithmetischer Mittelwert . . . . .	37
5.1.2	Exponentielle Glättung . . . . .	37
5.1.3	Lineare Regression . . . . .	38
5.2	Netzwerkkommunikation . . . . .	39

5.3	Zeitsynchronisation . . . . .	40
5.3.1	NTP-Protokoll . . . . .	40
5.3.2	Grundlagen . . . . .	41
5.3.3	Implementierung . . . . .	41
5.3.4	Vorverarbeitung . . . . .	42
5.3.5	Drift und Regression . . . . .	42
5.4	Signalverarbeitung . . . . .	43
5.4.1	Geglätteter Drift . . . . .	46
5.4.2	Schwellwertfunktion . . . . .	47
5.4.3	Charakteristisches Signalmerkmal . . . . .	48
5.5	Experiment . . . . .	50
5.5.1	Präzision . . . . .	50
5.5.2	Exkurs: Messung der Schallgeschwindigkeit . . . . .	51
5.6	Soundkartenlatenz . . . . .	52
5.7	Weiterführende Signalverarbeitung . . . . .	54
5.8	Zusammenfassung . . . . .	55
<b>6</b>	<b>Komponentenintegration</b>	<b>56</b>
6.1	Offline-Experiment . . . . .	56
6.2	Live-Ortung mit bekannten Mikrofonpositionen . . . . .	59
6.2.1	Versuchsaufbau . . . . .	60
6.2.2	Ergebnis . . . . .	61
6.2.3	Diskussion . . . . .	62
6.3	Ortung mit unbekanntem Mikrofonpositionen . . . . .	63
<b>7</b>	<b>Ausblick</b>	<b>67</b>
7.1	Herausforderungen . . . . .	67
7.2	Portabilität . . . . .	68
	<b>Literaturverzeichnis</b>	<b>71</b>

# 1 Einleitung

Die Lokalisierung von Signalquellen ist eine der bedeutendsten Aufgaben in der modernen Navigation, bei der Geländeaufklärung oder in der mobilen Robotik. Mehrere Ansätze existieren, um diese Ortung durchzuführen. Ein verbreitetes Verfahren misst die Amplitude eines Signals *RSSI* (Received Signal Strength Indication) und berechnet approximativ die Entfernung zum Sender. Existieren mindestens drei dieser Distanzen, so lässt sich eine *Trilateration* durchführen. Ein prinzipieller Nachteil ist die Anfälligkeit für Verfälschungen durch Hindernisse in der Umgebung, insbesondere bei Indoor-Szenarien [1, 2].

*Multilaterale Verfahren* hingegen messen die Differenz der Signallaufzeit bei mehreren Empfängern, sie werden auch *TDOA*-Verfahren (Time Difference Of Arrival) genannt. Sie sind nur von der Signalgeschwindigkeit abhängig, die im Allgemeinen von der Umgebungsbeschaffenheit nicht, oder nur wenig, beeinflusst wird.

Der Begriff der Multilateration kann leicht verwechselt werden mit der *Trilateration* oder der *Triangulation* [3]. Bei der Trilateration wird die Position eines Empfängers mit Hilfe von drei oder mehr Distanzmessungen zu Signalsendern ermittelt. Jeder Sender schränkt die Position des Empfängers auf (im zweidimensionalen Fall) einen Kreis ein, die Schnittpunkte dreier Kreise ergeben eine eindeutige Position. Bei der Triangulation ist eine Grundlinie bekannt und von ihren Endpunkten aus zwei Winkel zu einem unbekanntem Objekt. Seine Position ergibt sich als dritter Punkt eines Dreiecks unter Anwendung des „WSW“-Kongruenzsatzes (bekannter Winkel, Seitenlänge, Winkel) [4].

## 1.1 Motivation

Die Ortung von Sendern oder Empfängern durch Multilateration ist ein gut erforschtes Gebiet. Eine Menge von Sendern mit bekannten Positionen ermöglicht die Ortung eines Empfängers mit unbekannter Position. Eine geläufige Anwendung ist das *Global Positioning System* und die GSM-Lokalisierung. Auf gleiche Weise erfolgt der umgekehrte Fall der Ortung einer Signalquelle mit bekannten Empfängern, auch hierzu existieren Anwendungen. In Abschnitt 2.1 wird ein studentisches Experiment vorgestellt, welches dieses Ziel verfolgt.

Allen Anwendungen ist gemein, dass entweder die Position der Sender oder die der Empfänger bekannt ist. Das Wissen um die Position kann durch manuelle Vermessung erlangt werden oder durch Berechnung mittels eines externen Ortungssystems. Die manuelle Bestimmung ist akzeptabel, wenn es sich um ein stationäres System handelt. Für mobile Geräte ist sie zu umständlich und daher nicht praktikabel. Die Position der Empfänger könnte durch ein Ortungssystem wie GPS ermittelt werden. Anschließend würde durch Multilateration die Position der Sender bestimmt – oder andersherum die der Empfänger.

In vielen Fällen ist es jedoch wünschenswert, dass ein mobiles oder stationäres Sensornetz in der Lage ist, eine autarke Positionsbestimmung durchzuführen:

- Wenn die **Präzision** vorhandener Systeme nicht ausreicht: Für Anwendungen in Innenräumen ist GPS zu ungenau.
- Wenn vorhandene Systeme räumlich oder temporär nicht **verfügbar** sind: Unter der Erde oder unter Wasser ist ein Ortungssystem wie GPS nicht zu empfangen. Aufgrund einer Krise oder einer Katastrophe könnten etablierte Systeme ausfallen.
- Die **Kosten** für externe Transceiver könnten deutlich höher als für primitive Sender und Empfänger sein, wie sie für TDOA-Verfahren benötigt werden.
- Aus politischen oder wirtschaftlichen Gründen könnte eine **Abhängigkeit** von einem übergeordneten Ortungssystem zu vermeiden sein.

Ein experimentelles Anwendungsszenario, welches uns spontan in den Sinn kam, war die Trittschallortung im Rechnerpool der Universität mit den Rechnern als Empfängern. Dieses Experiment könnte eine Vorbereitung für ein mobiles Raumüberwachungssystem sein. Eine weitere Idee ist die Ortung eines Gewitters, indem mit einem Server synchronisierte Mobiltelefone Donnerschläge aufnehmen und die Zeitpunkte an den zentralen Server weiterleiten.

Allgemein ist die Lokalisierung mobiler Sensoren innerhalb eines Netzwerks interessant. Mit dem Wissen um die relative Positionierung eines Sensorknotens im Netz könnte beispielsweise das Routing von Information verbessert werden. Hierfür würde man keine Information über die absolute Position im Raum benötigen, die relative Position zu den anderen Knoten würde genügen. Anschaulich wird das am Beispiel des Roboterfußballs: Möchte ein Roboter einen Pass zu einem Mitspieler spielen, so benötigt er nicht dessen absolute Position auf dem Spielfeld, nur seine relative Position zu sich selbst im Netzwerk seiner Teamkollegen. Es stellt sich die zentrale Frage dieser Arbeit:

*Ist eine laufzeitbasierte Lokalisierung von Sensoren ohne gegebene Positionen der Gegenparte möglich?*

Es ist leicht ersichtlich, dass auf diese Weise keine absolute Positionsbestimmung durchführbar ist: Es sind keinerlei Positionen von Objekten vorgegeben, damit existieren keine Bezugspunkte im Raum. So wird die Ausrichtung, bzw. Orientierung eines Sensornetzes im Raum unmöglich. Häufig ist das auch nicht notwendig, etwa im beschriebenen Beispiel des Roboterfußballs: Für einen Pass reicht es aus, die relative Position des Mitspielers zu kennen.

In dieser Arbeit wird untersucht, ob eine solche Positionsbestimmung möglich ist und welche Voraussetzungen gegeben sein müssen. Mit diesem Wissen wird ein Verfahren entwickelt, mit dem sich die laufzeitbasierte, wechselseitige Lokalisierung von Schallquellen und Empfängern (Mikrofonen) vornehmen lässt.

## 1.2 Gliederung

Diese Arbeit ist in mehrere Teile gegliedert, welche in den folgenden Kapiteln dargelegt werden. Zunächst wird ein Überblick über thematisch verwandte Arbeiten gegeben.

Danach werden in Kapitel 3 allgemeine Fragen der laufzeitbasierten Lokalisierung beleuchtet. Von besonderer Bedeutung ist die Frage der Zuordnung aufgenommener Schallsignale zu logischen Schallquellen und die eindeutige Lösbarkeit eines Szenarios in Abhängigkeit von der Objektanzahl (Freiheitsgrad). Anschließend wird der Ansatz zur Berechnung der wechselseitigen Lokalisierung erläutert, der eigens entwickelte *Lampshade*-Algorithmus (Kapitel 4). Nacheinander werden die geometrische Repräsentation der Objekte, die darauf ausgeführte physikalische Simulation, die Programmierparadigmen und anschließend eine experimentelle Bestätigung der Hypothesen beschrieben.

Nach dem simulatorischen Ansatz wird in Kapitel 5 das *BasicTimestamping*-Modul geschildert. Es wurde unabhängig von der Simulation entwickelt und ermöglicht die Signalaufzeichnung an Laptops und die Verwendung der aufgezeichneten Zeitstempel als Eingangsdaten für den *Lampshade*-Algorithmus. Es wird beschrieben, wie die Laptops im Netz kommunizieren, wie sie ihre internen Uhren synchronisieren und wie die Signalverarbeitung abläuft. Ein experimenteller Teil überprüft die Plausibilität und Präzision des Timestampings.

Im dritten Teil (Kapitel 6) folgt die Integration von *Lampshade*-Algorithmus und *Timestamping*-Modul. Experimentell wird untersucht, inwiefern unbekannte Schallquellen mit den entwickelten Komponenten geortet werden können.

Ein Ausblick vervollständigt die Arbeit und gibt eine Übersicht über noch offene Fragen. Er erläutert, wie die weitere Vorgehensweise aussehen könnte und inwiefern sich die bestehenden Ansätze auf mobiler Hardware umsetzen lassen.

## 2 Themenverwandte Arbeiten

Im Gegensatz zur Ortung mit unbekanntem Mikrofonpositionen existieren zur Frage der laufzeitbasierten Ortung mit bekanntem Objektpositionen mehrere Ansätze. Sie alle haben gemein, dass entweder die Position der Sender oder die der Empfänger gegeben ist.

Das wohl bedeutendste, aktive System ist das satellitengestützte *Global Positioning System* [4], welches in der Bachelorarbeit des Autors [5] bereits angesprochen wurde. Ein Netz von nicht-geostationären Satelliten sendet permanent Funksignale mit Position und Uhrzeit zur Erde. Empfängt ein GPS-Receiver mindestens vier dieser Signale, so kann er seine Position durch Bestimmung der Laufzeitunterschiede auf bis zu 10 Meter genau bestimmen.

Auch die Lokalisierung einer Schallquelle mit vier Mikrofonen durch Banerji und Pande [6] wurde bereits in der Bachelorarbeit des Autors [5] diskutiert. Die Arbeit ist jedoch nach wie vor von Interesse, da sie als Referenz für die Schallortung im geschlossenen Raum dient.

### 2.1 Sound Source Triangulation Game

Im Rahmen eines Seminars für Microcontroller Design an der Cornell University in Ithaca, New York, entwickelten die beiden Studenten Bohnish Banerji und Sidharth Pande im Frühjahr 2007 eine praktische Arbeit zur laufzeitbasierten Schallortung mit vier Mikrofonen.

Ziel des „Sound Source Triangulation Game“ [6] war die Ortung eines Probanden, welcher nach Aufforderung des Programms in die Hände klatscht. Die vier an bekannten Positionen im Raum platzierten Mikrofone zeichneten das Signal auf und ein *Atmel Mega32*-Microcontroller führte einen Newton-Algorithmus aus, der anhand der generierten Zeitstempel die eindeutige Position des Probanden ermittelte.

Die zu lösende Gleichung beschreibt die Ausbreitung der Schallwelle im Raum in Abhängigkeit von der Schallgeschwindigkeit  $c$ , den Empfangszeitpunkten  $t_i$  an jedem Mikrofon und den Mikrofonpositionen  $x_i$ ,  $y_i$  und  $z_i$  ( $i = 1...4$ ):

$$f_i(x, y, z, t) = c(t - t_i) - \sqrt{(x - x_i)^2 + (y - y_i)^2 + (z - z_i)^2} \quad (2.1)$$

Der in der Programmiersprache C implementierte Algorithmus iteriert über eine Startposition  $\vec{x}_0$  und löst das aus Gleichung 2.1 entstehende polynomielle Gleichungssystem:

$$\vec{b} = A\Delta\vec{x} \quad (2.2)$$

$$\begin{bmatrix} -f_1(x_0, y_0, z_0, t_0) \\ -f_2(x_0, y_0, z_0, t_0) \\ -f_3(x_0, y_0, z_0, t_0) \\ -f_4(x_0, y_0, z_0, t_0) \end{bmatrix} = \begin{bmatrix} \frac{df_1}{dx} & \frac{df_1}{dy} & \frac{df_1}{dz} & \frac{df_1}{dt} \\ \frac{df_2}{dx} & \frac{df_2}{dy} & \frac{df_2}{dz} & \frac{df_2}{dt} \\ \frac{df_3}{dx} & \frac{df_3}{dy} & \frac{df_3}{dz} & \frac{df_3}{dt} \\ \frac{df_4}{dx} & \frac{df_4}{dy} & \frac{df_4}{dz} & \frac{df_4}{dt} \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta z \\ \Delta t \end{bmatrix} \quad (2.3)$$

$\vec{b}$  ist der Vektor der Funktionswerte der Startposition,  $\Delta\vec{x}$  enthält das Update jeder Iteration und  $A$  ist die Jacobi-Matrix, die die Ableitung nach jeder Dimension enthält. Die Elemente von  $A$  entstehen aus:

$$\begin{aligned} \frac{df_n}{dx} &= \frac{x - x_n}{\sqrt{(x - x_n)^2 + (y - y_n)^2 + (z - z_n)^2}} \\ \frac{df_n}{dy} &= \frac{y - y_n}{\sqrt{(x - x_n)^2 + (y - y_n)^2 + (z - z_n)^2}} \\ \frac{df_n}{dz} &= \frac{z - z_n}{\sqrt{(x - x_n)^2 + (y - y_n)^2 + (z - z_n)^2}} \\ \frac{df_n}{dt} &= c \end{aligned}$$

Zur Berechnung des Updatevektor für jede Iteration wird  $A$  invertiert. Es wird iteriert, bis  $\Delta\vec{x}$  hinreichend klein ist und das Verfahren abbricht.

$$\Delta\vec{x} = A^{-1}\vec{b} \quad (2.4)$$

Die vom Proband erzeugten Schallimpulse werden durch einen Schmitt-Trigger detektiert und an einen I/O-Port des Mikrocontrollers weitergeleitet. Es wird keine sampleweise Verarbeitung des Signals durchgeführt, was in Anbetracht der geringen, verfügbaren Rechenleistung sinnvoll erscheint.

Mit diesen Daten sollte sich über den Laufzeitunterschied des Signals eine eindeutige Position berechnen lassen. Nach Angaben der Autoren funktionierte das leider nicht wie erwünscht. Die Ursache hierfür konnte nicht endgültig geklärt werden. Ein Grund für den Fehlschlag schien die Genauigkeit des Timestamping mit mehreren Millisekunden Abweichung zu sein. Erwähnt wurde auch ein Problem beim Invertieren der Jacobi-Matrix.

Die gestellte Aufgabe der Schallortung im geschlossenen Raum ist nicht einfach gewählt. Die geringen Entfernungen stellen hohe Anforderungen an die Präzision der Signalerfassung. Erschwerend kommen Störungen durch Reflexionen an den Wänden hinzu. In Kapitel 6 werden eigene Versuche durchgeführt, die bessere Ergebnisse erbrachten, allerdings mit anderer Hardware. Auch das experimentelle Setup von vier

Mikrofonen in Tetraeder-Anordnung wurde nachgebaut. Die Ergebnisse sind implizit in den vorgestellten Experimenten enthalten und werden deshalb nicht gesondert aufgeführt.

## 2.2 Robust Sound Source Localization

Einen analytischen Ansatz stellen Valin et al. [7] vor. Wie in der vorigen Arbeit wird ein statisches Netz aus bekannten Mikrofonen im Raum installiert, die Art der lokalisierten Schallsignale und der mathematische Ansatz zur Lösung hingegen unterscheiden sich.

Auf einem mobilen Roboter wird ein Array von 8 Mikrofonen in einer Kastenform der Dimensionen  $50\text{ cm} \times 40\text{ cm} \times 36\text{ cm}$  befestigt. Eine externe, in einem Desktop-Rechner installierte Multikanal-Soundkarte nimmt die Signale entgegen. Auf diesem Rechner wird eine Kreuzkorrelation der Kanäle mit inverser Fouriertransformation berechnet. Dadurch erhalten Valin et al. eine Korrelation bei einer bestimmten Verschiebung der Signale gegeneinander. Eine amplitudenabhängige Gewichtung mit dem Ziel, den Beitrag des Rauschens zu vermindern, verstärkt die Korrelation des Nutzsignals. Nach Angaben der Autoren kann so relativ robust der zeitliche Offset breitbandiger Signale bestimmt werden. Schmalbandige Signale, insbesondere Sinustöne, seien weniger geeignet. Der Laufzeitunterschied zwischen je zwei Mikrofonen ergibt sich aus der Verschiebung mit der maximalen Korrelation, geteilt durch die Schallgeschwindigkeit  $c$ .

Mit diesen Laufzeitunterschieden  $\Delta T_{ij}$  wird unter der Annahme eines weit vom Array entfernten Schallsignals eine Winkelapproximation durchgeführt, wobei

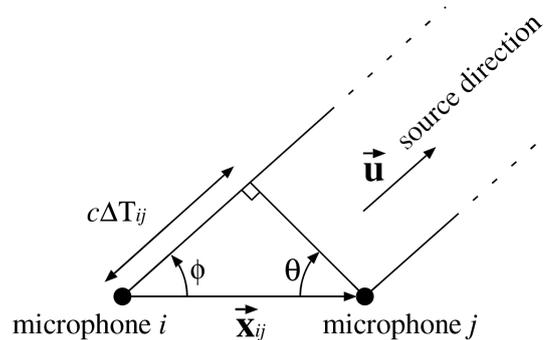
$$\cos \phi = \frac{\vec{u} \cdot \vec{x}_{ij}}{\|\vec{u}\| \|\vec{x}_{ij}\|} = \frac{\vec{u} \cdot \vec{x}_{ij}}{\|\vec{x}_{ij}\|} \quad (2.5)$$

$$\cos \phi = \sin \theta = \frac{c\Delta T_{ij}}{\|\vec{x}_{ij}\|} \quad (2.6)$$

$$(2.5) + (2.6) \Rightarrow \vec{u} \cdot \vec{x}_{ij} = c\Delta T_{ij} \quad (2.7)$$

Der Einheitsvektor  $\vec{u}$  gibt die Richtung der Schallquelle an, vgl. Abb. 2.1. Aus den Vektoren  $\vec{x}_{ij}$  für alle Paarungen von Mikrofonen ergibt sich ein lineares Gleichungssystem, welches nach  $\vec{u}$  aufgelöst wird. Die aus  $\vec{x}_{ij}$  entstehende Matrix ist invertierbar, solange sich nicht alle Mikrofone an derselben Stelle befinden.

Für Schallquellen in endlicher Entfernung ergibt sich durch die Approximation eine Parallaxe, ein Winkelfehler für nahe gelegene Schallquellen. Die Ergebnisse von Valin et al. lassen jedoch vermuten, dass er ab einer Entfernung oberhalb von einem Meter eine untergeordnete Rolle spielt. Auf diese Weise können Valin et al. sehr präzise den Winkel, nicht jedoch die Entfernung einer Schallquelle bestimmen.



**Abbildung 2.1:** Robust Sound Source Localization: Berechnung des Richtungsvektors  $\vec{u}$  unter Verwendung des Laufzeitunterschieds zweier Mikrofone  $i$  und  $j$ . [7]

## 2.3 Stat-Bat

Ein weiterer Ansatz zur laufzeitbasierten, wechselseitigen Lokalisierung ist der von Christian Schindelbauer entworfene *Stat-Bat*-Ansatz. Auch er kommt ohne iterative Verfahren aus, die Positionen der Signalsender und -empfänger werden analytisch berechnet. Dazu nutzt er eine trigonometrische Approximation über mindestens zwei Laufzeitunterschiede von drei Schallquellen zu zwei Mikrofonpaaren. Die Vorgehensweise ähnelt der Positionsschätzung durch Valin et al. [7]. Auch hier wird der Kosinus des Winkels zu einer weit entfernten Schallquelle unter Annahme unendlich weit entfernter Schallquellen approximiert. Auf diese Weise wird der Winkel der Schallquelle zur Grundlinie der Mikrofonpaare bestimmt. Im Gegensatz zu Valin et al. ist jedoch die Länge der Grundlinie, der Abstand der Mikrofone untereinander, nicht bekannt. In der Bachelorarbeit von Natascha Widder [8] wird der formale Unterbau des Ansatzes erläutert.

Ein eigener Test mit künstlich generierten Daten zeigte, dass das Verfahren grundsätzlich funktioniert. Drei Mikrofone wurden auf einem gleichseitigen Dreieck mit 4 Metern Kantenlänge positioniert, eine Serie von 100 Szenarien mit je drei Schallquellen wurde auf mehreren Orbits um das Zentrum des Dreiecks zufällig platziert. Die detaillierte Aufstellung der Ergebnisse verlässt den Fokus dieser Arbeit und wird deshalb übersprungen. Zusammengefasst lassen sich aber folgende Punkte feststellen:

- Sind die Schallquellen signifikant außerhalb des Perimeters der Mikrofone, so nimmt der Approximationsfehler von  $\alpha$ ,  $d_1$  und  $d_2$  ab und erreicht niedrige Werte. Für Schallquellen innerhalb des Perimeters der Mikrofone scheint die Ortung nicht geeignet, da  $d_1$  und  $d_2$  zu groß werden.
- Entfernungsabhängig schlugen 5 – 20% der Ortungen entfernter Schallquellen fehl. Möglicherweise liegt dies an Singularitäten bei der Schallquellenpositionierung. In den Orbits auf Höhe der Mikrofone erreichte die Zahl der Fehlschläge

ein Maximum. Darüber nahmen die Fehlschläge proportional zu den Approximationsfehlern ab.

- Auch mit einem Fehlermodell mit 1 ms normalverteiltem Laufzeitfehler konnten plausible Ergebnisse erzielt werden. Bei 4 ms waren die Ergebnisse hingegen unbrauchbar. Dies ist bei der geringen Setupgröße von 4 Metern Kantenlänge nicht überraschend.

Der Approximation innewohnend ist ein Fehler, der für weit entfernte Schallquellen minimal ist, und für nähere Schallquellen zunimmt. Es stellt sich die Frage, für welche Szenarien man den Ansatz verwenden kann und für welche er aufgrund des Fehlers ungeeignet ist. Einerseits zeigen die Ergebnisse, dass bestimmte Mindestabstände nicht zu unterschreiten sind. Andererseits nimmt der Fehler einer positionalen Ortung mit steigender Entfernung der Schallquelle zu: Betrachtet man das Dreieck zwischen Schallquelle und zwei Mikrofonen, so führt ein minimaler Fehler bei der Messung des Laufzeitunterschieds zu einem umso größeren Positionsfehler, je spitzer der Winkel des Dreiecks an der Schallquelle ist. Dies könnte zu einer Maximalentfernung für eine sinnvolle Ortung führen, die selbstverständlich für alle laufzeitbasierten Algorithmen gilt.

Sollte man ein geeignetes Distanzfenster finden, so könnte die Stat-Bat-Technik dazu verwendet werden, eine grobe Voranalyse der Mikrofonpositionen zu liefern. Diese würden mit dem hier vorgestellten Algorithmus (Kapitel 4) verfeinert werden. Vorteile wären sowohl die Vermeidung lokaler Minima (s. Abschnitt 4.7), als auch eine Beschleunigung der Iteration durch Auswahl günstiger Startpositionen.

## 3 Lokalisierung

In einem  $p$ -dimensionalen, euklidischen Raum seien je eine Menge von  $m$  Mikrofonen und  $n$  Schallquellen gegeben. Ein Mikrofon sei ein Tupel  $(x_1, \dots, x_p, [t_1, \dots, t_n])$ , eine Schallquelle sei  $(x_1, \dots, x_p, t)$ . Ihre jeweiligen Positionen  $x$  sowie der Zeitpunkt  $t$  seien unbekannt. Bekannt ist nur eine Menge von Zeitpunkten („Timestamps“) für jedes Mikrofon, zu denen es Signale von den Schallquellen erhält. Ein Schallsignal sei so definiert, dass man ihm einen exakten Zeitpunkt zuordnen kann, etwa, indem es von sehr kurzer Dauer ist. Unter Verwendung der Signalzeitpunkte an den Mikrofonen und der Signalgeschwindigkeit  $c$  kann man Rückschlüsse über die Positionen ziehen: *Inwiefern lassen sich die Positionen der Empfänger und Sender mit den gegebenen Informationen bestimmen?*

Für jedes Mikrofon und jede Schallquelle existiert eine Menge unbekannter Variablen, ihre Position im Raum und ein Ereigniszeitpunkt  $t$ . Die Signallaufzeit zwischen einem Mikrofon  $M$  und einer Schallquelle  $S$  beschränkt diese Variablen und bildet *Constraints* unter der Verwendung der Formel:

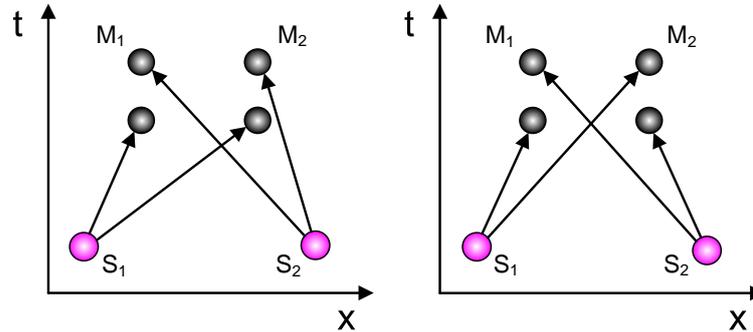
$$c \cdot (t_M - t_S) = \sqrt{(x_{M1} - x_{S1})^2 + (x_{M2} - x_{S2})^2 + \dots + (x_{Mp} - x_{Sp})^2} \quad (3.1)$$

Constraints verringern den Freiheitsgrad der möglichen Positionen  $x$ . In Abschnitt 4.2 werden sie ausführlich beschrieben.

### 3.1 Assoziationsproblem

Empfängt ein Mikrofon ein Schallsignal, so ist zunächst nicht klar, von welcher Schallquelle es stammt. Existiert nur eine einzige Schallquelle, oder sind die Schallsignale zeitlich weit genug voneinander entfernt, so ist die Zuordnung von Signal zu erzeugtem Timestamp auf mehreren Aufnahmegeräten eindeutig. Im Allgemeinen ist das jedoch nicht der Fall, wie in Abbildung 3.1 illustriert ist.

Eine Menge von  $n$  Schallquellen kann auf  $n!$  verschiedene Arten mit den maximal  $n$  Timestamps eines Mikrofons verknüpft sein, alle Permutationen sind zu berücksichtigen. Diese Zuordnung wird für alle  $m$  Mikrofone wiederholt. Die Schallquellen sind nicht unterscheidbar. Aus diesem Grunde ist bei genau einem Mikrofon nur die nicht-permutierte Identität zu betrachten. Es ergeben sich  $(n!)^{m-1}$  mögliche Assoziationen.



**Abbildung 3.1:** 2 Schallquellen und 2 Mikrofone, die je zwei Signale empfangen. Es ergeben sich zwei mögliche Assoziationen: [12] [12] (links) und [12] [21] (rechts). Letztere ist wahrscheinlicher.

Schallquellen	Mikrofone					
	1	2	3	4	5	6
1	1	1	1	1	1	1
2	1	2	4	8	16	32
3	1	6	36	216	1296	7776
4	1	24	576	13824	331776	7962624
5	1	120	14400	1728000	2.07E+8	2.49E+10
6	1	720	518400	3.73E+8	2.69E+11	1.93E+14

**Tabelle 3.1:** Die Anzahl der Möglichkeiten,  $n$  Schallquellen mit  $m$  Mikrofonen zu verknüpfen ist  $(n!)^{m-1}$ . Bereits ab drei Mikrofonen können nicht mehr alle Möglichkeiten durchlaufen werden.

Die Zuordnung kann man beispielsweise in Form eines Strings aus  $m$  Tupeln zu je  $n$  Einträgen angeben. Der Eintrag an der ersten Stelle im ersten Tupel bezeichnet die Verknüpfung der ersten Schallquelle mit einem Timestamp des ersten Mikrofons, der zweite Eintrag im ersten Tupel die zweite Schallquelle mit einem anderen Timestamp des ersten Mikrofons, usw. Die Strings lassen sich sortieren und von der ersten Permutation (Identität) bis zur letzten aufzählen:

$$\underbrace{[12\dots n] [12\dots n] \dots [12\dots n]}_{m \text{ mal}} \quad \text{bis} \quad [12\dots n] [n\dots 21] \dots [n\dots 21]$$

Die Zahl der Möglichkeiten wächst mit zunehmender Objektzahl sehr schnell, siehe Tabelle 3.1. Es ist im Allgemeinen nicht möglich, alle Verknüpfungen zu probieren und die mit dem besten Ergebnis zu verwenden. In der entwickelten Simulation wird vorausgesetzt, dass die korrekte Assoziation bekannt ist – als Zuordnungs-String wird die Identität übergeben. Das ist erlaubt, wenn man davon ausgeht, dass zwischen zwei Signalen ein ausreichender, zeitlicher Abstand ist.

Schallquellen	Mikrofone									
	1	2	3	4	5	6	7	8	9	10
1	0	2	4	6	8	10	12	14	16	18
2	3	4	5	6	7	8	9	10	11	12
3	6	6	6	6	6	6	6	6	6	6
4	9	8	7	6	5	4	3	2	1	0
5	12	10	8	6	4	2	0	-2	-4	-6
6	15	12	9	6	3	0	-3	-6	-9	-12
7	18	14	10	6	2	-2	-6	-10	-14	-18
8	21	16	11	6	1	-4	-9	-14	-19	-24
9	24	18	12	6	0	-6	-12	-18	-24	-30
10	27	20	13	6	-1	-8	-15	-22	-29	-36

**Tabelle 3.2:** 3D-Lokalisierung mit unbekanntem Positionen: Anzahl notwendiger Objekte, damit das Gleichungssystem eindeutig wird (nicht-positiver Freiheitsgrad). Ab 4 Schallquellen bzw. 5 Mikrofonen lässt sich eine eindeutige Lösung finden.

Die in Kapitel 5 vorgestellte Timestamping-Software hält eine Grenze von ca. 100 ms zwischen Timestamps ein, nicht zuletzt, weil etwa ein Händeklatschen bereits eine Dauer von 100 ms hat. Damit ist die Zuordnung in einem Raum der Größenordnung von 30 Metern eindeutig. Ohne eine solche Grenze müsste man sich spätestens bei der Umsetzung der realen Schallortung Gedanken zu dieser Frage machen. Heuristische Verfahren könnten unplausible Zuordnungen im Vorfeld ausschließen, von den Schallcharakteristika könnten Invarianten berechnet werden, oder man könnte eine Frequenzanalyse oder eine Korrelation zweier Signale durchführen. In jedem Fall müssten die Signale rechenaufwändig getrennt und zugeordnet werden, was in Abschnitt 5.7 noch einmal angesprochen wird.

## 3.2 Freiheitsgrade

Die Zahl der Schallquellen und Mikrofone hat maßgeblichen Einfluss darauf, ob das numerische Lösungsverfahren gegen ein eindeutiges Szenario iteriert. Das Gleichungssystem entsteht aus der Menge unbekannter Variablen, die es zu bestimmen gilt, und der Gleichungen (*Constraints*), die die Zahl der möglichen Lösungen eingrenzen. Nimmt die Zahl der Unbekannten zu, so erhöht sich der Freiheitsgrad  $G$  des Systems. Ein Gleichungssystem mit dem Freiheitsgrad  $G = 0$  ist eindeutig bestimmt. Ist der Freiheitsgrad kleiner als 0, so ist es überbestimmt und nur lösbar, wenn sich die Constraints nicht widersprechen.

Die Frage ist, ob das Gleichungssystem ab einer bestimmten Anzahl von Objekten eindeutig lösbar ist, und wie groß diese Zahl ist. Wir betrachten den allgemeinen Fall

Schallquellen	Mikrofone									
	1	2	3	4	5	6	7	8	9	10
1	1	2	3	4	5	6	7	8	9	10
2	3	3	3	3	3	3	3	3	3	3
3	5	4	3	2	1	<b>0</b>	<b>-1</b>	<b>-2</b>	<b>-3</b>	<b>-4</b>
4	7	5	3	1	<b>-1</b>	<b>-3</b>	<b>-5</b>	<b>-7</b>	<b>-9</b>	<b>-11</b>
5	9	6	3	<b>0</b>	<b>-3</b>	<b>-6</b>	<b>-9</b>	<b>-12</b>	<b>-15</b>	<b>-18</b>
6	11	7	3	<b>-1</b>	<b>-5</b>	<b>-9</b>	<b>-13</b>	<b>-17</b>	<b>-21</b>	<b>-25</b>
7	13	8	3	<b>-2</b>	<b>-7</b>	<b>-12</b>	<b>-17</b>	<b>-22</b>	<b>-27</b>	<b>-32</b>
8	15	9	3	<b>-3</b>	<b>-9</b>	<b>-15</b>	<b>-21</b>	<b>-27</b>	<b>-33</b>	<b>-39</b>
9	17	10	3	<b>-4</b>	<b>-11</b>	<b>-18</b>	<b>-25</b>	<b>-32</b>	<b>-39</b>	<b>-46</b>
10	19	11	3	<b>-5</b>	<b>-13</b>	<b>-21</b>	<b>-29</b>	<b>-37</b>	<b>-45</b>	<b>-53</b>

**Tabelle 3.3:** 2D-Lokalisierung: Anzahl notwendiger Objekte, damit das Gleichungssystem eindeutig wird. Hier reichen 3 Schallquellen bzw. 4 Mikrofone für einen Freiheitsgrad von  $G \leq 0$  aus.

der 3D-Lokalisierung mit  $n$  unbekanntem Schallquellen und  $m$  unbekanntem Mikrofonen. Sie bilden eine 3D-Struktur im Raum. Bei den Mikrofonen sind die Zeitpunkte bekannt, zu denen sie ein Signal erhalten. Im 3D-Raum hat eine Schallquelle die vier Unbekannten  $x, y, z$  und  $t$ . Ein Mikrofon hat einen unbekanntem Ort  $x, y, z$  und eine Menge bekannter Timestamps  $T$ . Die Timestamps der Mikrofone sind von denen der Schallquellen abhängig, deshalb werden sie nicht gezählt. Desweiteren ist jede Schallquelle mit je einem Timestamp mit allen Mikrofonen assoziiert, wir erhalten  $mn$  Constraints, die wir abziehen. Die absolute Lage und Orientierung der Struktur im Raum interessiert uns nicht. Sie ließe sich ohne eine gegebene Position ohnehin nicht bestimmen. Für Translation und Rotation der Struktur kann man deshalb je 3 Freiheitsgrade abziehen. Wir erhalten:

$$G_{3D} = 4n + 3m - nm - 6 \tag{3.2}$$

Wie man in Tabelle 3.2 sehen kann, ist das Gleichungssystem für kleine  $m$  und  $n$  ( $m, n < 4$ ) unterbestimmt. Sind aber genügend Objekte jedes Typs verfügbar, so lässt sich Formel 3.2 zufolge eine Lösung finden. Ab 4 Schallquellen bzw. ab 5 Mikrofonen ist das Gleichungssystem eindeutig, genügend Objekte der anderen Menge vorausgesetzt. In der Praxis ist die Zahl der benötigten Objekte unter Umständen höher. Bilden vier Mikrofone oder Schallquellen eine Ebene, so erhöht das vierte Objekt den Informationsgehalt der Konfiguration nicht weiter, was im folgenden Abschnitt erläutert wird.

Im Falle der 2D-Lokalisierung besitzt ein Objekt je eine Raumkoordinate weniger. Die Translation und Rotation lassen zwei, bzw. eine Unbekannte wegfallen. Wir erhalten:

$$G_{2D} = 3n + 2m - nm - 3 \tag{3.3}$$

Eine Hypothese dieser Arbeit ist, dass sich künstliche oder reale Szenarien mit einem Freiheitsgrad von  $G < 0$  eindeutig bestimmen lassen<sup>1</sup>. Dies ist ab einer bestimmten Mindestanzahl von Mikrofonen und Schallquellen der Fall. Bei einigen Grenzfällen kann die Bestimmbarkeit durch massive Erhöhung der Mikrofon- bzw. Schallquellenanzahl noch erreicht werden, z.B. im 3D-Fall mit 4 Schallquellen (Tabelle 3.2).

### 3.3 Qualität der Verteilung im Raum

Die angegebenen Freiheitsgrade stellen nur den bestmöglichen Fall dar, also das erzielbare Minimum der Freiheitsgrade. Liegen die Objekte in ungünstiger Konstellation vor, etwa einer Ebene, so nimmt die Anzahl effektiv raumaufspannender Objekte ab.

Beispiel: Zur Ortung einer Schallquelle genügen im allgemeinen Fall vier bekannte Mikrofone. Liegen diese jedoch in einer Ebene, so lässt sich diese Ebene durch drei Mikrofone beschreiben und ein Mikrofon wird redundant, seine Positionsinformation geht verloren. Die Schallquelle kann sich nun auf einer Parabel (1 Freiheitsgrad) befinden, anstatt nur auf einem Punkt, siehe auch Wendeborg, 2007 [5], Abb. 3.2.

Nähern sich die Positionen einer Objektgruppe einer beliebigen Ebene an, so verschlechtert sich die Lösbarkeit des Szenarios bereits. Man kann diese Eigenschaft numerisch angeben, indem man die minimale Standardabweichung des Abstandes der Objekte von der Ebene berechnet. Eine Ebene ( $\mathbf{N}, d$ ) lässt sich durch den Normalenvektor  $\mathbf{N}$  und den Abstand vom Ursprung  $d$  beschreiben. Im Folgenden werden Vektoren und Matrizen immer in fettgedruckter Notation dargestellt. Die Dimension ergibt sich jeweils aus dem Kontext.

Die Ebene zu finden, die von allen Objekten den minimalen Abstand hat, geht einher mit dem Optimierungsproblem, die minimale Standardabweichung des Abstands von der Ebene zu bestimmen. Mit linearer Regression kann die Ebene bestimmt werden, zu der die quadratischen Abstände der Objekte minimal sind. Der mittlere quadratische Abstand von dieser Ebene dient als Qualitätsmerkmal für die initiale Schallquellen- und Mikrofonverteilung. Im Folgenden wird die Methode der Regression erläutert.

#### 3.3.1 Multiple Regression

Mit der linearen Regression lässt sich nicht nur eine lineare Messwertentwicklung nachverfolgen, wie es beispielsweise bei der Zeitsynchronisation des Timestamping geschieht (Kapitel 5). Es ist auch möglich, eine mehrdimensionale Ebene zu finden, die durch eine Partikelmenge gegeben ist. Wir verwenden hierzu die *multiple linear Regression* [3, 4].

<sup>1</sup>Der Fall  $G = 0$  sei ausgeschlossen, da hierzu die Mikrofon- und Schallquellenpositionen ideal sein müssen.

Die deskriptive Menge  $\mathbf{X}$  sei mehrdimensional mit  $n$  Dimensionen. Zusammen mit  $m$  Messwerten schreiben wir eine  $m \times (n + 1)$ -Matrix mit Einsen in der ersten Spalte. Die Beobachtung  $\mathbf{Y}$  hat die Dimension  $m \times 1$ . Wir führen einen Regressionsvektor  $\beta$  ein. Er enthält den „y-Achsenabschnitt“  $\beta_0$  und die „Steigungen“  $\beta_i$  ( $0 \leq i \leq n$ ) und beschreibt die Ebene, hat also die Dimension  $(n + 1) \times 1$ . Mit dem Vektor der Residuen  $\epsilon$  erhalten wir das Gleichungssystem in Matrixschreibweise:

$$\mathbf{Y} = \mathbf{X}\beta + \epsilon \quad (3.4)$$

Die Methode der kleinsten Quadrate minimiert  $\epsilon$  durch die Umschreibung:

$$e_i = y_i - \hat{\beta}_0 - \hat{\beta}_1 x_1 - \dots - \hat{\beta}_n x_n \quad (3.5)$$

Für alle  $m$  Eingabewerte erhalten wir:

$$\sum_{i=1}^m \sum_{k=1}^n \mathbf{X}_{ij} \mathbf{X}_{ik} \hat{\beta}_k = \sum_{i=1}^m \mathbf{X}_{ij} \mathbf{Y}_i, \quad (j = 1, \dots, n) \quad (3.6)$$

In Matrixschreibweise lösen wir auf nach  $\hat{\beta}$ :

$$\hat{\mathbf{Y}} = (\mathbf{X}^T \mathbf{X}) \hat{\beta} = \mathbf{X}^T \mathbf{Y} \quad (3.7)$$

$$\Rightarrow \hat{\beta} = (\mathbf{X}^T \mathbf{X})^{-1} (\mathbf{X}^T \mathbf{Y}) \quad (3.8)$$

Durch Verwendung einer geeigneten Vektorbibliothek lässt sich der implementatorische Aufwand für die Invertierung von  $(\mathbf{X}^T \mathbf{X})$  klein halten. Die C++-Library *Eigen* [9] bietet die Funktion `Matrix::inverse()` mit guter numerischer Stabilität dank Pivotisierung, die sich auf alle invertierbaren Matrizen anwenden lässt. Es ist ratsam, doppelte Fließkommapräzision zu verwenden.

### 3.3.2 Abstand von der Ebene

Das Ergebnis der oben beschriebenen Regression ist der Vektor  $\hat{\beta} = (b_0, b_1, \dots, b_n)$ , der den y-Achsenabschnitt  $b_0$  und die Steigungen in alle Richtungen enthält. Zusammen mit den Basisvektoren der Raumdimensionen erhalten wir:

$$\mathbf{B} = \begin{bmatrix} 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 1 \\ b_0 & b_1 & b_2 & \dots & b_n \end{bmatrix} \quad (3.9)$$

wobei die Spaltenvektoren von  $\mathbf{B}$  die Ebene  $P(\mathbf{N}, d)$  aufspannen. Aus den Vektoren  $\mathbf{B}_1, \dots, \mathbf{B}_{n-1}$  lässt sich die Normale berechnen mit  $\mathbf{N} = \mathbf{B}_1 \times \mathbf{B}_2 \times \dots \times \mathbf{B}_{n-1}$ .

Die Hessesche Normalform beschreibt den Abstand eines Punktes  $\mathbf{P}$  von einer Ebene mit  $\mathbf{N} \cdot \mathbf{P} = d$ . Wir kennen den Spaltenvektor  $\mathbf{B}_0$  als Punkt auf der Ebene. Eingesetzt erhalten wir die Distanz  $d$  der Ebene zum Ursprung:

$$d = \mathbf{N} \cdot (\mathbf{B}_0 - \vec{0}) \tag{3.10}$$

Nun wird der mittlere Abstand aller Punkte von der gefundenen Ebene berechnet. Die Varianz des Abstands, welche mit Hilfe des Normalenvektors berechnet wird, weicht von der Quadratsumme der Residuen ab, denn diese ist abhängig von der Lage der Ebene. Der euklidische Abstand wurde daher als geeigneteres Maß angesehen und wird nochmal explizit berechnet.

### 3.3.3 Kreuzprodukt

Die Ebenenregression wurde sowohl im dreidimensionalen, als auch im vierdimensionalen Raum verwendet. Bei Ersterer können mögliche Ebenen, auf denen eine Objektgruppe liegt, in der Visualisierung durch geeignete Rotation der Darstellung einfach gesehen werden. Auch die Berechnung von  $\mathbf{N}$  ist unproblematisch, da das Kreuzprodukt „ $\times$ “ für den  $\mathbb{R}^3$  bekanntermaßen definiert ist.

Der vierdimensionale Raum hingegen lässt sich nicht visuell darstellen, zumal die  $t$ -Dimension im dargestellten Szenario komprimiert ist und durch Multiplikation mit der Schallgeschwindigkeit  $c$  zunächst entzerrt werden muss. Der Ebenenabstand lässt sich auf gleiche Weise berechnen, jedoch ist es notwendig, zur Ermittlung des Normalenvektors das Kreuzprodukt im  $\mathbb{R}^4$  einzuführen:

$$\mathbf{a}_1 \times \mathbf{a}_2 \times \mathbf{a}_3 = \begin{vmatrix} \mathbf{e}_1 & a_{11} & \dots & a_{13} \\ \mathbf{e}_2 & a_{21} & \dots & a_{23} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{e}_4 & a_{41} & \dots & a_{43} \end{vmatrix} \tag{3.11}$$

Die Determinante mit den Haupteinheitsvektoren  $\mathbf{e}_i$  ( $i = 1, \dots, 4$ ) kann durch den Laplaceschen Entwicklungssatz aufgelöst werden. Die resultierenden, knapp 80 Koeffizienten müssen einmal aufgeschrieben und in korrekter Weise multipliziert, addiert und subtrahiert werden. Der Test auf Korrektheit hingegen ist einfach: Für den resultierenden Vektor  $\mathbf{N} = \mathbf{a}_1 \times \mathbf{a}_2 \times \mathbf{a}_3$  gilt  $\mathbf{N} \cdot \mathbf{a}_i = 0$  für  $i = 1, \dots, 3$ , er steht also senkrecht auf allen  $\mathbf{a}_i$ .

## 4 Lampshade-Algorithmus

Der Schwerpunkt dieser Arbeit lag auf der Untersuchung, *ob* und *wie* sich die Frage unbekannter Schallquellen und Mikrofone mit einem iterativen Approximationsverfahren lösen lässt. Zunächst war unklar, ob der aus dem Constraint in Gleichung 3.1 gewonnene Richtungsvektor  $\mathbf{N}_0$  (folgender Abschnitt) ausreicht, um plausible Ergebnisse zu erhalten. Es bestand die Möglichkeit, dass zusätzlich eine Wechselwirkung der Mikrofone untereinander zu berücksichtigen ist. Beispielsweise kann man mit Hilfe der korrespondierenden Zeitstempel zweier Mikrofone etwas über ihre Maximaldistanz zueinander aussagen. Wichtiger als die Performanz war deshalb die maximale Flexibilität und Nachvollziehbarkeit der Iteration. Aus diesem Grunde wurde ein geometrisches Modell des Problems entwickelt und mittels einer physikalischen Simulation genähert. Auf diese Weise kann einfach untersucht werden, ob die laufzeitbasierte, wechselseitige Lokalisierung überhaupt lösbar ist, und wenn nicht, kann ein Grund dafür gefunden werden.

Das verwendete Modell zur Positionsbestimmung unbekannter Schallquellen und Mikrofone versucht, die Signallaufzeit  $\Delta t$  über die Distanz  $\Delta s$  mit  $c\Delta t = \Delta s$  zu relaxieren. Dabei ist  $\Delta s$  die euklidische Entfernung, sie enthält eine Wurzel. Die resultierenden polynomialen Gleichungssysteme lassen sich nicht analytisch lösen und werden häufig durch ein Iterationsverfahren numerisch approximiert. Das bekannteste unter ihnen ist das Newtonverfahren (Newton-Raphsonsche Methode).

### 4.1 Exkurs: Newtonsches Iterationsverfahren

In der Arbeit von Banerji und Pande [6] wurde das Newton-Verfahren eingesetzt, um bei vier bekannten Mikrofonen eine Schallquelle zu lokalisieren. Es galt, die Gleichung

$$\vec{b} = A\Delta\vec{x} \tag{4.1}$$

mit Funktionenvektor  $\vec{b}$ , Jacobi-Matrix  $A$  und Updatevektor  $\Delta\vec{x}$  nach  $\Delta\vec{x}$  aufzulösen, s. Abschnitt 2.1. Die Funktionsfähigkeit konnte in einem Nachbau durch den Autor im Herbst 2007 [5] bestätigt werden. Während dieser Experimente bestach das Newtonverfahren durch seine schnelle Konvergenz. Bis zum Erreichen der Abbruchbedingung wurden weniger als ein Dutzend Iterationen benötigt.

Eine Erweiterung des Algorithmus auf  $m$  bekannte Mikrofone wurde 2008 durch Falko Stenzel im Rahmen einer Studienarbeit [10] durchgeführt.  $n$  Schallquellen konnten bei gegebener Assoziation durch  $n$ -fache Ausführung des Iterationsverfahrens gefunden werden. Jede Schallquelle wird mit den jeweils korrespondierenden Timestamps separat gesucht, indem der Algorithmus jedes mal erneut gestartet wird. Das ist korrekt und sinnvoll, denn die Schallquellen sind nicht voneinander abhängig und nur durch die (gegebenen) Mikrofonpositionen bestimmt. Die Jacobi-Matrix  $A$  der Größe  $m \times 4$  ist, im Gegensatz zur Referenz von Banerji und Pande, nicht mehr quadratisch, damit existiert keine Inverse von  $A$  [11]. Der Autor löst das Problem durch eine Regressionsanalyse, der sogenannten „Methode der kleinsten Quadrate“ mit dem Gauß’schen Eliminationsverfahren mit Pivotisierung:

$$A^T \vec{b} = A^T A \Delta \vec{x} \quad (4.2)$$

$$\Delta \vec{x} = (A^T A)^{-1} (A^T \vec{b}) \quad (4.3)$$

Für die wechselseitige Ortung hingegen, also die Lösung eines Szenarios, dessen Mikrofonpositionen nicht bekannt sind, konnten keine Quellen gefunden werden, die eine Lösung mit dem Newtonverfahren untersuchen. Im betrachteten Fall der 3D-Ortung mit  $m$  Mikrofonen und  $n$  Schallquellen sind  $m + n$  Positionsvektoren mit 3 bzw. 4 Koordinaten  $(x, y, z, t)$  unbekannt. Die Positionen der Mikrofone und Schallquellen sind jeweils voneinander abhängig. Das Newtonverfahren muss alle  $3m + 4n$  Unbekannten gemeinsam lösen: Die Länge des Updatevektors  $\Delta \vec{x}$  wächst von 4 auf  $3m + 4n$ . Die Zahl der Constraints ist nicht mehr  $m$ , sondern  $mn$ , damit wächst  $\vec{b}$  auf  $mn \times 1$ . Die Größe der Matrix  $A$ , ist damit  $mn \times (3m + 4n)$ .

Gängige Algorithmen zur Invertierung einer Matrix bzw. Lösung eines Gleichungssystems, wie das Gaußverfahren, benötigen eine Laufzeit in  $\Theta(n^3)$ . Für große Szenarien wird das Newtonverfahren also rechenintensiver pro Iterationsschritt. Vermutlich steigt bei diesen komplexen Szenarien auch die Zahl der Schritte. Desweiteren muss sorgfältig darauf geachtet werden, dass die numerische Stabilität erhalten bleibt. Geeigneter als das Gauß’sche Eliminationsverfahren ist eine QR-Zerlegung mit Householdertransformationen [4].

Das newtonsche Iterationsverfahren gilt als schnell, da es quadratisch konvergiert und kann beinahe für beliebige, nichtlineare Gleichungen eingesetzt werden. Die Invertierung der Jacobi-Matrix bei der Lösung mehrdimensionaler Probleme ist jedoch rechenaufwändig und bei unterbestimmten Gleichungssystemen ist die Lösung willkürlich. Der vorgestellte *Lampshade*-Algorithmus hingegen besitzt eine Methode, um unterbestimmte Gleichungen sinnvoll zu lösen, siehe Abschnitt 4.3. Auch das Verhalten des Newtonverfahrens bei fehlerhafter Konvergenz aufgrund lokaler Minima ist nur schwer nachvollziehbar. Wie sich zeigen wird, können die federbasierten Kräftevektoren des *Lampshade*-Algorithmus einfach nachverfolgt werden, um lokalen Minima auf die Spur zu kommen, siehe Abschnitt 4.7.

## 4.2 Geometrische Darstellung

Wir betrachten wieder den  $n$ -dimensionalen Raum  $\mathbb{R}^n$ . In diesem sei eine Signalquelle  $S$  an der Position  $\mathbf{s}_S$  gegeben,  $\mathbf{s}_S$  ist  $n$ -dimensional. Zu einem Zeitpunkt  $t_S$  sendet sie ein Signal aus. Die Wellenfront des Signals breitet sich nach dem physikalischen Gesetz  $s = vt$  kugelförmig aus, für  $v$  setzen wir die Signalgeschwindigkeit  $c$  ein. Die Front hat also zum Zeitpunkt  $t_S + \Delta t$  die Entfernung  $d = c\Delta t$  von der Position der Schallquelle, bzw. die Kugel um  $S$  hat den Radius  $d$ .

Nun führen wir einen Empfänger dieses Signals  $M$  (Mikrofon) an der Position  $\mathbf{s}_M$  ein. Es erhält das Signal zum Zeitpunkt  $t_M = t_S + \Delta t$ . Die Wellenfront des Signals erreicht den Empfänger zum Zeitpunkt

$$t_W := t_S + \frac{\|\mathbf{s}_M - \mathbf{s}_S\|}{c} \quad (4.4)$$

Ein *gültiges* Mikrofon sei definiert als ein Mikrofon, dessen Zeitpunkt des Signalempfangs gleich dem Moment ist, in dem die Wellenfront des Signals es erreicht, also  $t_W = t_M$ .

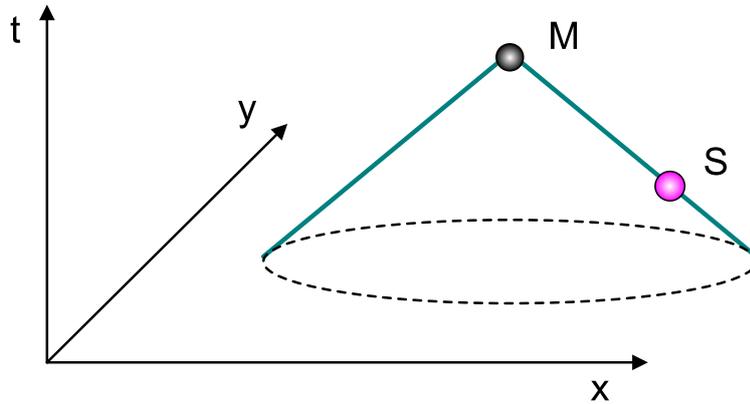
Wir nehmen eine zusätzliche Dimension  $t$  hinzu und erhalten so den Raum  $\mathbb{R}^{n+1}$ . In dieser tragen wir die Wellenfront zu jedem Zeitpunkt  $t'$  auf. Wir erhalten einen  $n+1$ -dimensionalen Kegel mit der Schallquelle als Spitze und der Steigung  $1/c$ . Nach obiger Definition gilt für jedes  $t' > t_S$ , dass  $t'_M = t_S + \Delta t' = t'_W$ , damit liegt das (gültige) Mikrofon auf dem Mantel des Kegels. Außerhalb des Kegels würde gelten  $t_M < t_W$ , der Empfänger hat das Signal schon gemessen, obwohl die Wellenfront ihn noch nicht erreicht hat, innerhalb des Kegels wäre es umgekehrt. Analog zu dieser Darstellung kann der Kegel umgedreht werden und von  $M$  ausgehen. Dann empfängt das Mikrofon  $M$  eine Schallquelle, deren unbekannte Position auf dem Kegelmantel liegt. Diese Darstellung ist für unsere Anwendung zweckmäßiger.

Zum besseren Verständnis des geometrischen Modells kann das Beispiel der zweidimensionalen Schallortung herangezogen werden. Wir können die Objekte als *Partikel* an einer bestimmten räumlichen und zeitlichen Position visualisieren. Auf diese Weise lässt sich das Modell sehr intuitiv darstellen. Das Schallsignal bildet einen Kreis mit dem Mikrofon-Partikel  $M$  als Zentrum. Auf die Zeitachse aufgetragen ergibt sich ein Kegel, siehe Abb. 4.1.

Wir rechnen ab jetzt häufig mit dem  $n+1$ -dimensionalen Positionsvektor von  $S$  und  $M$  und erweitern deshalb den Ortsvektor zu:

$$\mathbf{p}_S = [\mathbf{s}_S, t_S] \quad (4.5)$$

$$\mathbf{p}_M = [\mathbf{s}_M, t_M] \quad (4.6)$$



**Abbildung 4.1:** Zweidimensionale Ortung im dreidimensionalen Raum  $(x, y, t)$ : Das Mikrofon  $M$  empfängt ein Schallsignal  $S$ , welches auf dem Mantel eines Kegels unterhalb von  $M$  liegen muss.

Für ein Schallsignal, das abseits eines Kegelmantels liegt, kann ein Fehler  $E$  angegeben werden, siehe Abb. 4.2 *oben*. Die zugehörige Fehlerfunktion sei  $\Phi$ :

$$E = \Phi(\mathbf{p}_M - \mathbf{p}_S) = c(t_M - t_S) - \|\mathbf{s}_M - \mathbf{s}_S\| \quad (4.7)$$

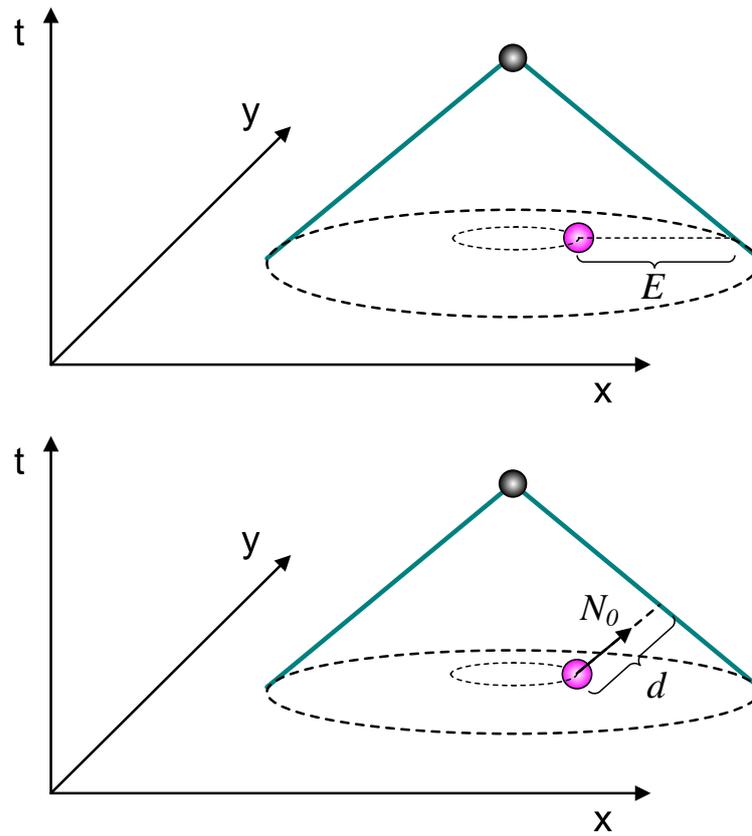
Gleichzeitig wird der günstigste, normalisierte Richtungsvektor  $\mathbf{N}_0$  berechnet, der die Gültigkeit des Mikrofons wiederherstellt. Er zeigt in Raumrichtung direkt aus dem „Kreis“ heraus auf den Kegel der Steigung  $-1/c$ , siehe Abb. 4.2 *unten*. In  $t$ -Richtung wird nicht der senkrechte Vektor auf den Kegel mit  $t$ -Steigung  $c$  verwendet, obwohl er intuitiv als am kürzesten erscheinen könnte. Der Grund ist, dass der Raum  $(x_1, x_2, \dots, x_n, t)$  nicht normiert ist, sondern in  $t$ -Dimension gestaucht ist. Eine Einheit in Richtung  $t$  entspricht  $c$  Einheiten in Richtung  $x_i$ . Wir verwenden zur Kompensation die  $t$ -Steigung  $1/c$ . In einem normierten Raum stünde  $\mathbf{N}_0$  tatsächlich senkrecht auf dem Kegel. Der Richtungsvektor setzt sich auf diese Weise aus der normalisierten Raumkomponente und der Zeitkomponente zusammen:

$$\mathbf{N}_0 = \left[ \frac{\mathbf{s}_S - \mathbf{s}_M}{\|\mathbf{s}_S - \mathbf{s}_M\|}, \frac{1}{c} \right] \quad (4.8)$$

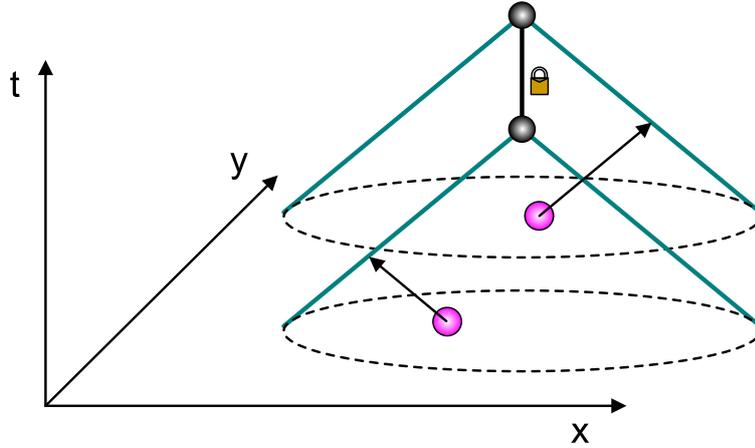
Für den Fall, dass  $t_S$  soviel kleiner ist als  $t_M$ , dass die Gerade in Richtung  $\mathbf{N}_0$  den Kegel nicht schneidet, was bei schlecht gewählter Startposition vorkommen kann, zeigt  $\mathbf{N}_0$  direkt in Richtung  $t$ , also  $\mathbf{N}_0 = [0, 0, \dots, \frac{1}{c}]$ . Das stellt einen Schnittpunkt mit dem Kegel sicher.

Multipliziert mit einem unbekanntem Faktor  $d$  erhalten wir einen Punkt auf dem Kegelmantel, für den gilt

$$\Phi(\mathbf{p}_M + d\mathbf{N}_0 - \mathbf{p}_S) = 0 \quad (4.9)$$



**Abbildung 4.2:** *Oben:* Der Fehler  $E$ , falls die Schallquelle  $S$  (pink) nicht auf dem Kegelmantel liegt. *Unten:* Der Richtungsvektor  $N_0$ , welcher dem günstigsten Weg zum Kegelmantel entspricht.  $d$  ist die Distanz zum Mantel. Wäre der Raum normiert, so stünde  $N_0$  senkrecht auf dem Kegel.



**Abbildung 4.3:** Ein Mikrofon empfängt zwei Schallsignale. Somit besitzt es zwei Zeitmarken und zwei zeitliche Positionen. Von der Lampenschirm-ähnlichen Staffelung der Kegel stammt der Name des Algorithmus.

Sicherlich ließe sich der Faktor  $d$  geometrisch berechnen, was sich als nicht ganz einfach erwies. Eine Entwicklung dieser Formel konnte eingespart werden: Da die Fehlerfunktion  $\Phi$  linear ist, kann sie mit Hilfe des Strahlensatzes an zwei beliebigen Punkten berechnet werden und auf diese Weise der Punkt ermittelt werden, an dem  $\Phi = 0$  ist. Wir erhalten:

$$d = \frac{\Phi(\mathbf{p}_M - \mathbf{p}_S)}{\Phi(\mathbf{p}_M - \mathbf{p}_S) - \Phi(\mathbf{p}_M + \mathbf{N}_0 - \mathbf{p}_S)} \quad (4.10)$$

Für jedes einzelne Schallsignal von jeder Schallquelle wird für das Mikrofon eine Zeitmarke generiert. Ein Mikrofon-Partikel befindet sich also an einer räumlichen und an mehreren zeitlichen Positionen. Es ergibt sich eine Staffelung von Mikrofon-Partikeln, jedes mit einem eigenen Kegel, siehe Abb. 4.3. Die Form dieser Staffelung erinnert an Lampenschirme einer Stehlampe. Sie gab dem *Lampshade*-Algorithmus seinen Namen.

Die auf diesem geometrischen Modell ausgeführte iterative Simulation hat zur Aufgabe, die Positionen von  $n$  Schallquellen-Partikeln und  $m$  Mikrofon-Partikeln so zu beeinflussen, dass der Gesamtfehler  $E_{\text{sum}}$  für jedes assoziierte Paar von Schallquelle und empfangenem Schallsignal minimiert wird:

$$E_{\text{sum}} \leftarrow \min_{\mathbf{p}_M, \mathbf{p}_S} \sum_{i=1}^m \sum_{j=1}^n \Phi(\mathbf{p}_{Mi} - \mathbf{p}_{Sj}) \quad (4.11)$$

Im Idealfall lässt sich ein Szenario finden, in dem jede Schallquelle *gültig* ist, d.h. sich auf den Kegelmanteln aller Mikrofone befindet.

### 4.3 Physikalische Simulation

Zur Suche nach *gültigen* Szenarien wird ein Verfahren verwendet, das auf einer physikalischen Masse-Feder-Simulation beruht. Die Inspiration des Autors hierzu kam aus der sehr guten Erfahrung mit Physiksimulationen, die während der Entwicklung von *HapticBillard* [12] im Winter 2008/09 gemacht wurden.

Grundlegend für diese Simulation ist die Einführung von Masseteilchen, welche den Newtonschen Trägheitsgesetzen gehorchen. Sie besitzen eine Position  $\mathbf{x}_t$ , eine Geschwindigkeit  $\mathbf{v}_t$  und eine Masse  $m$ . Ihre lineare Bewegung ändern sie nur, wenn eine Kraft  $\mathbf{F}_t$  auf sie einwirkt. Auch diese Kraft folgt physikalischen Grundsätzen. Verwendet werden hauptsächlich distanzabhängige Feder-Kräfte: Je größer die Distanz zur Neutrallänge, desto größer die Kraft. Um Schwingungen zu vermeiden und zur Stabilisierung der Simulation, wird eine Dämpfung appliziert. Die verwendete quadratische Dämpfung entspricht in ihrem Charakteristikum dem Luftwiderstand von in Atmosphäre bewegten Objekten.

Die zeitliche Integration wird durch ein einfaches Euler-Cromer-Verfahren mit einem Zeitschritt von  $h = 1$  ms umgesetzt:

$$\mathbf{x}_{t+h} = \mathbf{x}_t + h\mathbf{v}_{t+h} \quad (4.12)$$

$$\mathbf{v}_{t+h} = \mathbf{v}_t + h\frac{1}{m}\mathbf{F}_t \quad (4.13)$$

Versuchsweise wurde eine Runge-Kutta-4-Integration implementiert. Sie berechnet die Kräfte viermal pro Integrationsschritt und erreicht größere Genauigkeit und damit verbesserte Stabilität der Simulation. Größere Kräfte und ein größerer Zeitschritt wurden möglich, jedoch auf Kosten einer viermal größeren Rechenlast pro Integrationsschritt, was die Vorteile zunichte machte.

Die Kräfte wurden durch Federgleichungen simuliert, die die Partikel in Richtung der im vorigen Abschnitt beschriebenen Kegelmäntel ziehen. Sie können zu jedem Zeitpunkt der Simulation hinzugefügt oder entfernt werden. Ihr Betrag ist proportional zur Distanz vom Kegelmantel, unter Verwendung einer Federkonstante, der „Federhärte“. Diese Federn entsprechen der Korrespondenz zwischen einer logischen Schallquelle und dem Timestamp eines Mikrofons, also einem empfangenen Signal. Möglich ist entweder eine vollständige Assoziation in Form einer bestimmten Permutation der möglichen Zuordnung zwischen Timestamp eines Mikrofons zu einer Schallquelle, oder teilweise Assoziationen. Ersteres ergibt sich bei der Simulation mit künstlich generierten Daten. Die teilweise Assoziation entsteht bei der Verwendung realer Daten, nämlich dann, wenn ein Mikrofon ein Schallsignal verpasst hat.

Es wurde eine Funktion implementiert, um unterbestimmte Schallquellen darzustellen. Deren Aufenthaltsort ist nicht punktuell, sondern liegt auf einer Parabel, oder sogar auf einem Rundkegel, je nach Grad der Unterbestimmung. Dieser Fall kann auch bei

ausreichender Anzahl, aber ungünstiger Anordnung der Mikrofone auftreten. Zur Verdeutlichung der Unterbestimmung wird eine Schallquelle nicht nur durch ein Partikel dargestellt, sondern durch eine Wolke von z.B. 10 Partikeln, welche sich schwach voneinander abstoßen. Ist der Aufenthaltsort eindeutig, so sammelt sich die Wolke um ein Zentrum. Andernfalls sorgt die gegenseitige Abstoßung für eine Ausbreitung in Richtung der übrig bleibenden Freiheitsgrade. Leider stört die Partikelwolke die Lokalisierung überbestimmter Szenarien. In diesen Fällen sollte die Zahl der Partikel pro Schallquelle auf eins reduziert werden.

Die physikalische Simulation ermöglicht, das Vorschreiten des Szenarios während der Iteration genau zu beobachten. Mittels einer in OpenGL entwickelten Visualisierung können die Positionen der Partikel und die herrschenden Kräfte grafisch dargestellt werden. Sie vereinfacht auch den Vergleich mit den *Ground-Truth*-Objekten, also den Objekten, deren Position im Raum durch manuelle Messung ermittelt wurde. Die Geschwindigkeit der Simulation ist mit einer Zeitrafferfunktion frei wählbar.

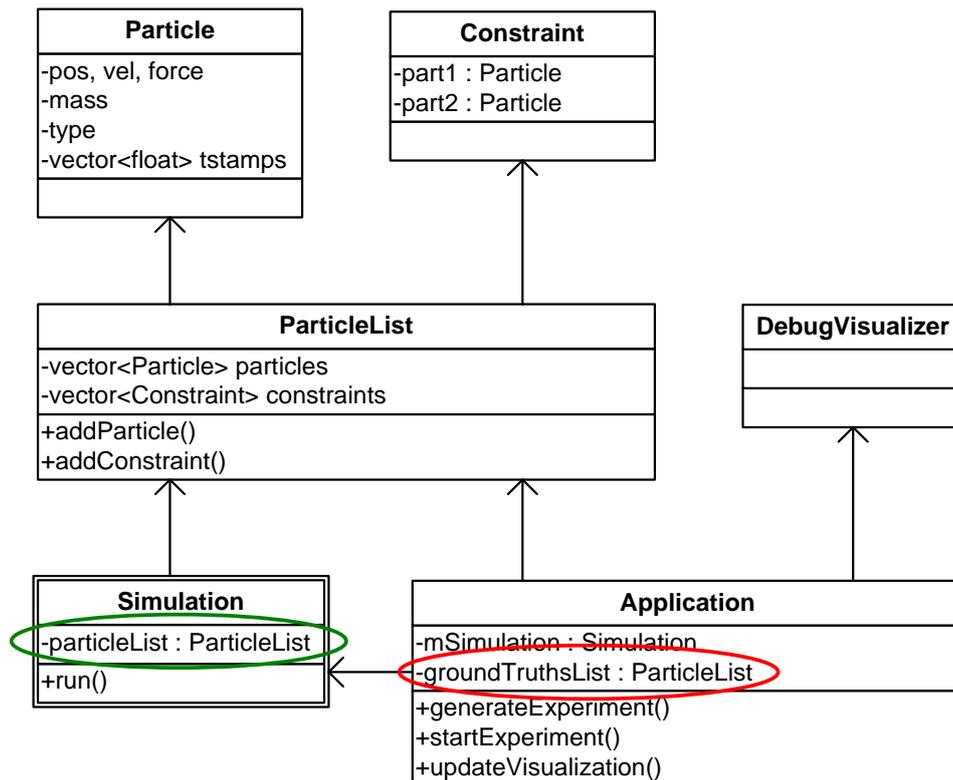
Diese Fähigkeiten sind mit einem klassischen Iterationsverfahren nur schwierig darstellbar, insbesondere das Problem der teilweisen Assoziation. Die Flexibilität dieser Simulation trug in Verbindung mit der grafischen Visualisierung erheblich dazu bei, Schwierigkeiten wie die lokalen Minima zu ergründen.

## 4.4 Programmierertechnik

Die Entwicklung eines Projekts birgt mit wachsender Codemenge zunehmend die Gefahr, in einen undurchsichtigen Dschungel aus Codezeilen zu geraten. Diese Erfahrung durfte der Autor bei dem frühen Projekt *Mics 'n Sounds* [5] machen. Auch das jüngste Projekt *HapticBillard* [12] mit seiner Größe von über 22.000 Codezeilen leidet zuweilen etwas an unübersichtlicher Strukturierung in den Verbindungsklassen.

Bei der Entwicklung der Lampshade-Software wurde Wert auf eine softwaretechnisch sinnvolle Strukturierung gelegt. Der entstehende Code sollte übersichtlich und erweiterbar, sicher und schnell sein:

- Die **Wartbarkeit** des Codes wird begünstigt durch eine thematische Gliederung in Objektklassen und Kapselung der Objekte. Diese objektorientierte Zugriffsbeschränkung vereinfacht die Zuweisung gültiger Daten. Indem der Code objektweise in einzelne Dateien aufgetrennt wird, können Codeelemente in verschiedenen Projekten wiederverwendet werden.
- Auf die Ausführungs-**Sicherheit** der Algorithmen wurde Rücksicht genommen, indem Vor- und Nachbedingungen von Funktionen sichergestellt werden. Gefährlicher Code wird durch Assertions und Exception Handling abgesichert, gemeinsame Objekte in verschiedenen Threads werden durch wechselseitigen Ausschluß (Mutex) geschützt, soweit möglich.



**Abbildung 4.4:** Vereinfachte Darstellung der Klassenabhängigkeiten. Sowohl die `Simulation`, als auch die `Application` greifen auf die Liste `Simulation::particleList` (grün umrandet) zu. Die `Simulation` hat keinen Zugriff auf die Ground-Truth-Daten in `Application::groundTruthsList` (rot umrandet).

- Maßgebend für die **Schnelligkeit** ist zwar die Laufzeitkomplexität und Datenmenge der Algorithmen, aber auch implementatorisch kann ein zügiger Ablauf des Codes begünstigt werden. Dazu gehört die Vermeidung wiederholter Berechnungen durch einmalige Vorberechnung (z.B. bei der z-Sortierung des DebugVisualizer), kein unnötiges Erzeugen und Zerstören von Objekten in Schleifen und eine sinnvolle Wahl von Datentypen.

Während der Testläufe muss die Software zwangsläufig Ground-Truth-Daten, also die realen Positionen der Partikel, erzeugen, aus denen ein Szenario mit Schallzeitpunkten (Timestamps) erstellt wird. Gleichzeitig darf die Simulation nur eine Teilmenge dieser Daten kennen, nämlich die Timestamps. Mit diesen Daten versucht sie, die Positionen zu rekonstruieren. Die Glaubwürdigkeit des Lampshade-Algorithmus wurde sichergestellt, indem eine logische Trennung der Datenbereiche vorgenommen wurde. So werden Positionsdaten in der Application-Klasse erzeugt, die Timestamps generiert, und die Positionen wieder auf Initialisierungswerte gesetzt. Erst anschließend wird die Simulation gestartet. Die Klasse Simulation hat niemals Zugriff auf die Ground-Truth-Daten, was in Diagramm 4.4 illustriert ist. Der Vergleich zwischen gefundenen und Ground-Truth-Daten erfolgt ausschließlich von der Application-Klasse aus.

## 4.5 Vergleich mit Ground-Truth

Das Ziel der wechselseitigen Lokalisierung ist die Bestimmung relativer Positionen und Abstände der Objekte untereinander. Es ist dann erreicht, wenn ein eindeutiges Ergebnis gefunden wurde, was gleichzeitig *gültig* ist, d.h. alle Schallquellen liegen auf den Kegelmänteln aller assoziierten Mikrofone.

Die absolute Raumposition der Objekte kann so nicht bestimmt werden. Das ist einleuchtend, denn der Simulation wurde beim Start, abgesehen von Zufallsvektoren um den Ursprung herum, keine einzige Raumkoordinate mitgegeben. Das gleiche gilt für die Orientierung des Netzwerks, auch sie kann nicht bestimmt werden.

Im Gegensatz zur Ortung mit *bekannt*en Mikrofonpositionen kann nicht direkt überprüft werden, ob sich die gefundene Schallquelle an derselben Position befindet, wie die im Experimental-Setup vorgegebene. Dort kann der Ortungsfehler durch Differenzbildung ermittelt werden, was bei unbekannten Mikrofonpositionen zunächst nicht möglich ist. Die wichtige Frage, ob das durch die Simulation gefundene Ergebnis dem Setup entspricht, muss dennoch geklärt werden.

Durch Drehung und Verschiebung des Netzes ändern sich die Abstände und relativen Positionen der Objekte zueinander nicht. Wir versuchen, eine orientierungserhaltende Rotation und eine Translation zu finden, die das Ergebnis der Simulation auf die Ground-Truth abbildet. Eine Methode, um die optimale Rotation zu bestimmen, ist die

Singulärwertzerlegung (SVD) [4], [13, Kap. 16] der Korrelationsmatrix beider Punktwolken  $X$  (Ground-Truth) und  $P$  (Experiment).

Zunächst eliminieren wir die räumliche Verschiebung (Translation) durch Bestimmung der Schwerpunkte  $\mu_x$  und  $\mu_p$  der jeweils  $n$  Schallquellen und  $m$  Mikrofone. Sie bilden sich aus den arithmetischen Mittelwerten (vgl. Abschnitt 5.1.1) der Ortsvektoren. Anschließend subtrahieren wir sie von beiden Punktwolken. Der neue Schwerpunkt liegt im Ursprung:

$$X' = \{\mathbf{x}'\} = \{\mathbf{x} - \mu_x\} \quad (4.14)$$

$$P' = \{\mathbf{p}'\} = \{\mathbf{p} - \mu_p\} \quad (4.15)$$

Nun bilden wir die Kreuzkorrelation  $\mathbf{W}$  durch Addition der dyadischen Produkte von  $X'$  und  $P'$ :

$$\mathbf{W} = \sum_{i=1}^{m+n} (\mathbf{x}'_i \mathbf{p}'_i^T) \quad (4.16)$$

Die Singulärwertzerlegung von  $\mathbf{W}$  sei die Zerlegung von  $\mathbf{W}$  in die Komponenten

$$\mathbf{W} = \mathbf{U} \mathbf{D} \mathbf{V}^T = \mathbf{U} \begin{bmatrix} \sigma_1 & 0 & 0 \\ 0 & \sigma_2 & 0 \\ 0 & 0 & \sigma_3 \end{bmatrix} \mathbf{V}^T \quad (4.17)$$

mit den Singulärwerten  $\sigma_i$  ( $1 \leq i \leq 3$ ) und den unitären Matrizen  $\mathbf{U}$  und  $\mathbf{V}$ . Ihr Produkt  $\mathbf{R} = \mathbf{U} \mathbf{V}^T$  bildet die Rotationsmatrix  $\mathbf{R}$ , die eine optimale Drehung der Punktwolke  $P$  auf  $X$  durchführt:

$$P'' = \{\mathbf{p}''\} = \{\mathbf{R}(\mathbf{p} - \mu_p) + \mu_x\} \approx \{\mathbf{x}\} = X \quad (4.18)$$

Der nach der Transformation verbleibende Fehler berechnet sich aus den quadrierten Abständen der einzelnen Punkte:

$$E_R = \sqrt{\frac{1}{m+n} \sum_{i=1}^{m+n} \|\mathbf{p}''_i - \mathbf{x}_i\|^2} \quad (4.19)$$

Realisiert wurde die SVD mit dem in der Vektorbibliothek *Eigen* [9] eingebauten Modul *SVD Solver*. Es verwendet vermutlich zunächst orthogonale Transformationen der Matrix  $\mathbf{W}$  und anschließend eine QR-Zerlegung [4], welche eine bessere numerische Stabilität aufweist, als die direkte Berechnung über die Eigenwerte von  $\mathbf{W}$ .

Anfangs war beabsichtigt, eine Rotationsmatrix  $\mathbf{R}$  zu finden, deren Determinante  $|\mathbf{R}| = 1$  ist, sie also *orientierungserhaltend* ist. Die SVD hingegen hat eine Diagonalmatrix  $\mathbf{D}$  mit nicht-negativen, realen Koeffizienten, ihre Determinante ist also nicht-negativ. Demzufolge muss die Determinante von  $\mathbf{U} \mathbf{V}^T$  dasselbe Vorzeichen wie  $\mathbf{W}$  haben. Wenn

bereits  $\mathbf{W}$  eine negative Determinante hatte, so muss auch  $\mathbf{UV}^T$  eine negative Determinante besitzen<sup>1</sup>. Weil  $\mathbf{R} = \mathbf{UV}^T$  unitär ist, d.h. ihre Spaltenvektoren orthonormal sind, ist in dem Fall  $|\mathbf{R}| = -1$ .

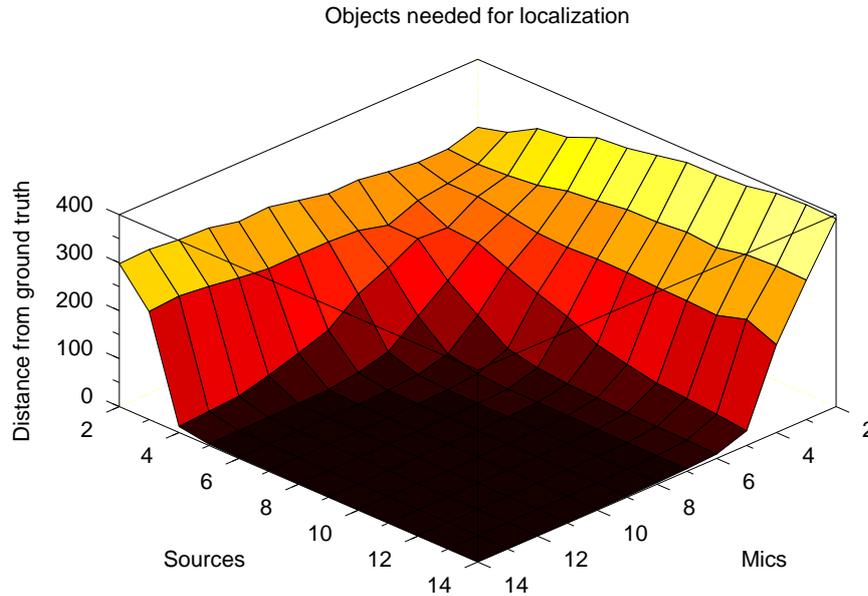
Was zunächst als Nachteil erschien, zeigte sich bald als große Bereicherung. Nicht in jedem Fall existiert eine sinnvolle, orientierungserhaltende Rotation von  $P'$  auf  $X'$ . Offenbar existiert ein Ergebnis der Simulation in Form einer Punktspiegelung, die (selbstverständlich) dieselben relativen Distanzen der Punkte aufweist, wie die Abbildung mit  $|\mathbf{R}| = 1$ . Iteriert die Simulation gegen diese punktgespiegelte Form der Ground-Truth, so würde die Rotation von  $X$  nach  $P$  mit der Determinante  $|\mathbf{R}| = 1$  einen großen Fehler ergeben. Somit liefert die SVD „automatisch“ die richtige Rotation mit  $|\mathbf{R}| = \pm 1$ , die  $P$  in  $X$  überführt. Abschnitt 4.7 beleuchtet den Sachverhalt der lokalen Minima genauer.

## 4.6 Experiment

Um die Leistungsfähigkeit des entwickelten Algorithmus zu testen, wurde ein Experiment mit künstlich erzeugten Zufallsszenarien durchgeführt. Das Ziel war herauszufinden, wieviele Schallquellen und Mikrofone für eine eindeutige Bestimmung notwendig sind, und den Prozentsatz zu bestimmen, wie viele Szenarien für jede Objektzahl *gültig* lokalisiert werden können, und wie viele in einem eventuellen lokalen Minimum landen. Zur Wiederholung sei gesagt, dass eine *gültige* Lokalisierung nicht automatisch eine erfolgreiche Lokalisierung bedeutet. Aufgrund von zu geringer Objektzahl, oder zu schlechter Objektverteilung, können Freiheitsgrade übrig bleiben. *Gültig* bedeutet lediglich die Eigenschaft, dass alle Schallquellen auf den Kegelmanteln der Mikrofone liegen (vgl. Abschnitt 4.2).

Mittels eines implementierten Fehlermodells kann ein normalverteilter Messfehler zum Empfangszeitpunkt hinzuaddiert werden. Ein Fehler von 1 ms bedeutet, dass der Empfangszeitpunkt mit 95,5%-iger Wahrscheinlichkeit um nicht mehr als 2 ms vom korrekten Zeitpunkt verschieden ist (gaußsche Glocke). Mit einem Fehler dieser Charakteristik und Größenordnung ist beim Timestamping, dem zweiten Teil dieser Arbeit, zu rechnen. Damit unterscheidet sich das Fehlermodell von dem, das in *Mics 'n Sounds* [5] zum Einsatz kam, und welches die Fehlergröße in Abhängigkeit von der Entfernung zwischen Sender und Empfänger berechnet, was unrealistisch ist. In diesem Experiment soll das Augenmerk jedoch auf der theoretischen Leistungsfähigkeit des Algorithmus liegen. Somit wurde das Experiment mit idealen Daten vorgenommen, also ohne einen künstlichen Messfehler des Empfangszeitpunktes am Mikrofon und ohne verlorene Timestamps (z.B. wenn das Mikrofon außer Reichweite des Schalls ist).

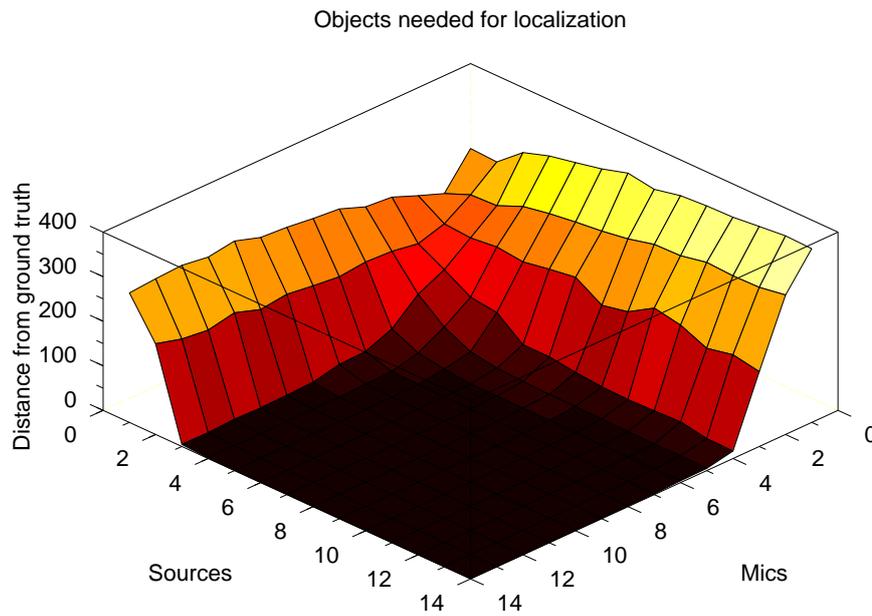
<sup>1</sup>Begründung einer Foreendiskussion auf <http://forum.kde.org/>, „Mathematical question: SVD“, Juli 2009 entnommen.



**Abbildung 4.5:** 3D-Lokalisierung, Experiment über 100 Zufallsszenarien, selektiert nach *gültigen* Versuchen. Dargestellt ist die Abweichung von der Ground-Truth für jede Kombination von Mikrofonzahl (2 – 14) zu Schallquellenzahl (2 – 14). Ab 4 Schallquellen bzw. 5 Mikrofonen kann eine erfolgreiche Ortung durchgeführt werden. Das Ergebnis deckt sich mit der Voraussage in Tabelle 3.2.

Mit Hilfe der verbleibenden Freiheitsgrade kann eine Abschätzung vorgenommen werden (vgl. Abschnitt 3.2): Sind sie kleiner oder gleich Null, so ist das zugrunde liegende Gleichungssystem eindeutig bestimmt bzw. überbestimmt, eine Ortung könnte möglich sein. Diese Aussage war experimentell zu belegen.

Es sollte herausgefunden werden, wieviele Objekte in einem simulierten Experiment für eine Ortung benötigt werden. Für jede Kombination von Mikrofonzahl zu Schallquellenzahl von 2 bis 14 wurden 100 Durchläufe mit zufällig generierten Positionen und Zeitpunkten gestartet, insgesamt  $13^2 \cdot 100 = 16.900$  Durchläufe im 3D-Fall. Im 2D-Fall wurden 1 bis 14 Objekte untersucht, insgesamt 19.600 Durchläufe. Der simulierte Raum hatte eine Größe von ca. 1000 Metern Kantenlänge, so dass der Schall ca. 3 Sekunden zur Durchquerung benötigte. Das 3D-Experiment dauerte ca. 4 Stunden, das 2D-Experiment erfolgte anschließend. Um die Dauer zu verkürzen, wurde es in drei Prozesse geteilt und einer auf einem 2-GHz-Athlon ausgeführt und zwei auf einem Core-Duo-Laptop mit 1,8 GHz und zwei Prozessorkernen.



**Abbildung 4.6:** 2D-Lokalisierung, Experiment über 100 Zufallsszenarien, selektiert nach *gültigen* Versuchen. Dargestellt ist die Abweichung von der Ground-Truth für jede Kombination von Mikrofonzahl (1 – 14) zu Schallquellenzahl (1 – 14). Ab 3 Schallquellen bzw. 4 Mikrofonen kann eine erfolgreiche Ortung durchgeführt werden. Das Ergebnis deckt sich mit der Voraussage in Tabelle 3.3.

Schallquellen	Mikrofone	Schleifendurchläufe	Zeitdauer in sec.
2	2	1.815	0,013
2	14	4.359	0,132
14	2	5.086	0,148
5	5	9.087	0,108
7	7	5.620	0,170
10	10	4.636	0,333
14	14	4.259	0,680

**Tabelle 4.1:** Durchschnittliche Laufzeit pro Ortung im 2D-Fall, gefiltert nach erfolgreichen Durchläufen (Auszug). Die Zeitdauer ist quadratisch zur Zahl der Objekte. Bemerkenswert ist die Abnahme der Schleifendurchläufe bei stark überbestimmten Szenarien.

### 4.6.1 Genauigkeit

Dank der SVD-Transformation ist der Vergleich der gefundenen Positionen mit der Ground-Truth einfach und man kann den RMS-Fehler zwischen beiden Objektwolken von Experiment und Ground Truth ausrechnen. Als Faustregel kann man einen RMS-Fehler von 100 Metern für alle Schallquellen als Schwellwert für eine erfolgreiche Ortung angeben. Die Ergebnisse sind bis auf die Grenzfälle der Lokalisierbarkeit eindeutig darüber oder darunter.

Betrachtet man die Diagramme 4.5 und 4.6, so stellt man fest, dass sich ihre Aussage genau mit der mittels der Freiheitsgrade getroffenen Voraussage in Abschnitt 3.2 deckt: Ab einer gewissen Anzahl von Objekten kann mit der entsprechenden Anzahl von Gegenobjekten eine eindeutige Lokalisierung stattfinden. Dies ist ab 4 Schallquellen bzw. 5 Mikrofonen im 3D-Szenario der Fall und ab 3 Schallquellen bzw. 4 Mikrofonen im 2D-Szenario.

### 4.6.2 Laufzeit

Im späteren 2D-Experiment war zwischenzeitlich die Auswertung verbessert worden, so dass die Zeitdauer der Durchläufe analysiert werden konnte. Sie ist von mehreren Faktoren abhängig. Eine überbestimmte Ortung, die in ein lokales Minimum läuft, also niemals *gültig* wird, erreicht nicht den geringen RMS-Fehler für eine Abbruchbedingung, sondern bricht erst ab, wenn die maximale Anzahl von 30.000 Schleifendurchläufen erreicht ist. Diese fehlgeschlagenen Durchläufe wurden zur Laufzeitanalyse weggelassen, da davon ausgegangen wird, dass sich die lokalen Minima in der Zukunft beseitigen lassen. Danach ist die Dauer quadratisch proportional zur Objektanzahl, wegen der quadratischen Anzahl von Constraints. Auf besagtem 2-GHz-PC ergaben sich im 2D-Fall die in Tabelle 4.1 abgebildeten, durchschnittlichen Laufzeiten.

Kriterium	Fehlgeschlagen	Erfolgreich
Anzahl	963	3.037
Distanz $d$ zwischen Partikelwolken	232,77 (112,11)	227,29 (119,13)
Radius Mikrofone $r_m$	304,61 (78,30)	327,35 (76,44)
Radius Schallquellen $r_s$	355,71 (55,75)	347,13 (57,51)
Quotient der Radien $r_m/r_s$	0,88 (0,28)	0,97 (0,29)
Quotient $d/((r_m + r_s)/2)$	0,73 (0,41)	0,70 (0,42)
Abstand von Regression, Mikrofone	501,01 (285,12)	469,60 (240,92)
Abstand von Regression, Schallquellen	466,32 (170,14)	461,21 (168,98)

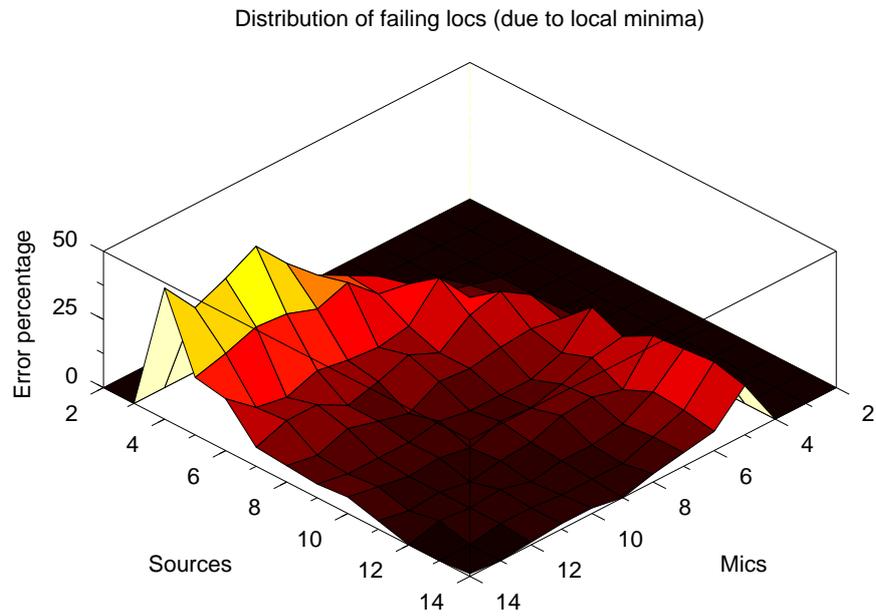
**Tabelle 4.2:** Untersuchung der zufälligen Partikelwolken auf bestimmte Kriterien hin. Die Werte geben den Durchschnitt über 963 bzw. 3.037 Messwerte an, die Werte in Klammern bezeichnen die Standardabweichung. Das 2D-Experiment wurde mit 4 Mikrofonen und 7 Schallquellen durchgeführt.

## 4.7 Lokale Minima

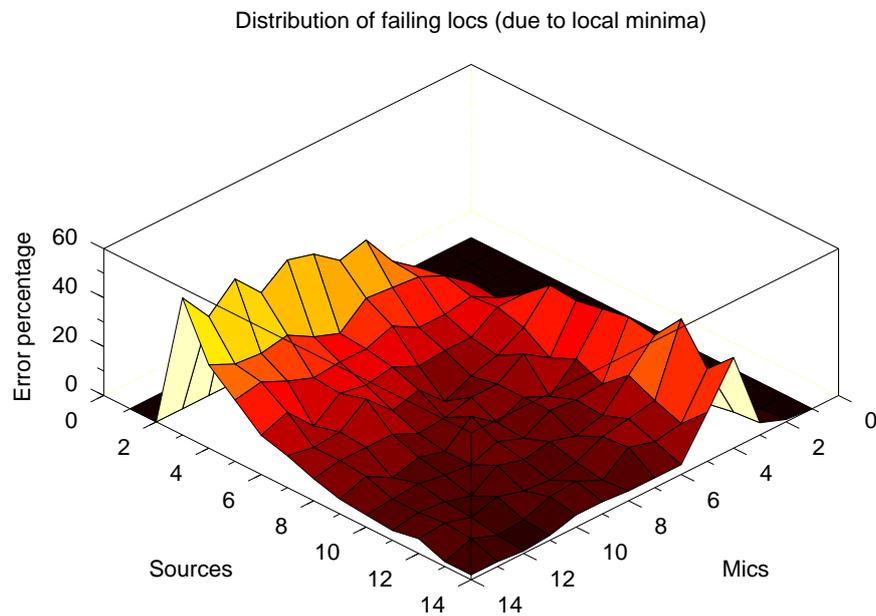
Es stellte sich heraus, dass der Algorithmus, insbesondere in Grenzfällen der Lokalisierbarkeit, in lokale Minima geriet, die eine *gültige* Lösung des Szenarios verhinderten. Sie stellen den Gegenpart zu den unterbestimmten Szenarien dar: In unterbestimmten Szenarien gibt es nicht genügend Objekte, oder Objekte mit zu wenig Informationsgehalt, z.B. kollineare Mikrofone, weswegen Freiheitsgrade verbleiben. Sie bilden eine ein- oder mehrdimensionale Lösungsmenge mit *gültigen* Lösungen, alle mit einem Fehler von  $E = 0$ . Hier ist es hilfreich, die in Abschnitt 3.3 erläuterte Prüfung des Ebenenabstandes durchzuführen. Ungeeignete Szenarien können so quantitativ angegeben werden.

Die lokalen Minima sind jedoch Folge einer eindeutigen Bestimmung, oder Überbestimmtheit des Szenarios, in der die optimale Lösung nicht gefunden werden kann. Auch hier wurde die Nähe bestimmter Partikel zu einer durch die Gegenpartikel definierten Ebene vermutet: Möglicherweise geraten Partikel auf die falsche Seite dieser Ebene, wo sie durch die Constraints blockiert werden. Ein Zusammenhang zwischen der Verteilung der Partikel und der Neigung zu lokalen Minima war naheliegend. Ein 2D-Experiment mit 4000 zufälligen Ortungen von 4 Mikrofonen und 7 Schallquellen wurde durchgeführt und in *erfolgreiche* und *fehlgeschlagene* Ortungen klassifiziert. Als fehlgeschlagen gelten alle Szenarien, in denen die Abbruchbedingung eines maximalen RMS-Fehlers von 1 Meter nicht erreicht werden konnte. In diesen Fällen lässt sich keine Rotation finden, die deckungsgleich zur Ground-Truth ist.

Anschließend wurde die Ground-Truth beider Klassen auf verschiedene Kriterien hin untersucht (Tabelle 4.2). Am wahrscheinlichsten erschien ein Zusammenhang zwischen dem Abstand der Partikel von einer Regressionsebene (hier: Gerade, da 2D-Raum). Hinsichtlich der hohen Standardabweichung, bedingt durch die zufällig generierten



**Abbildung 4.7:** 3D-Lokalisierung: Wegen lokaler Minima fehlgeschlagene Lokalisierungen in Prozent. Bemerkenswert ist der Kreis, dessen Rand an der Schwelle zur Lokalisierbarkeit liegt.



**Abbildung 4.8:** 2D-Lokalisierung: Wegen lokaler Minima fehlgeschlagene Lokalisierungen in Prozent. Bemerkenswert ist der Kreis, dessen Rand an der Schwelle zur Lokalisierbarkeit liegt.

Positionen, lässt sich jedoch keine signifikante Abweichung zwischen beiden Klassen feststellen. Auch die anderen Kriterien zeigen keinen deutlichen Unterschied. Das Experiment wurde mit 6 Mikrofonen und 6 Schallquellen wiederholt. Auch hier zeigen sich nur minimale Unterschiede bei großer Standardabweichung. Entweder wurde das ausschlaggebende Kriterium der Objektverteilung nicht gefunden, oder die Objektverteilung im Raum ist nicht ausschlaggebend für die Tendenz zu lokalen Minima.

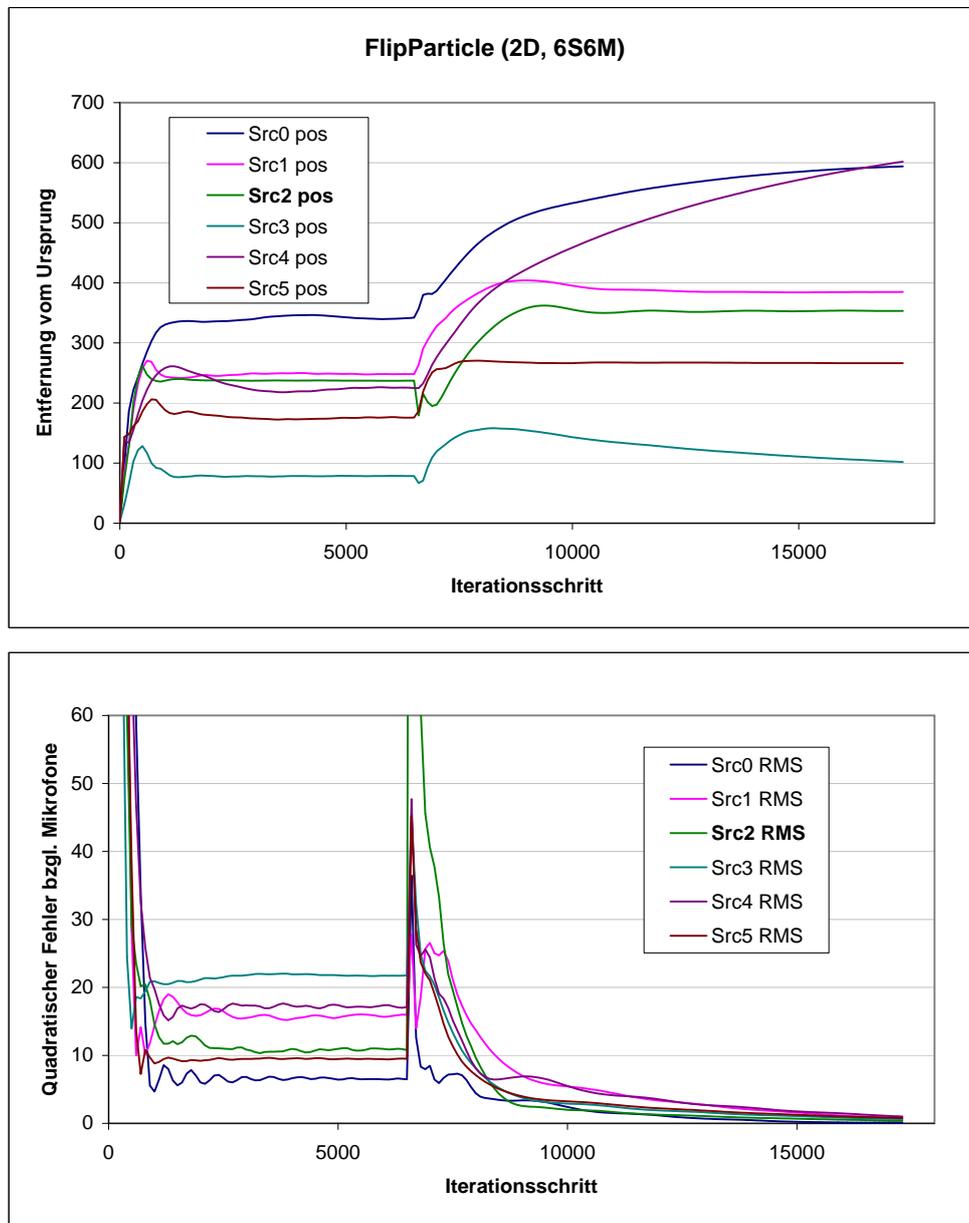
Im zweiten Experiment mit variierenden Objektzahlen wurde die Häufigkeit der Fehlschläge quantisiert. Es zeigte sich ein kreisrunder Gürtel im Diagramm, der in den schlechtesten Fällen bis zu 40 % aller Lokalisierungen vereitelte, s. Abb. 4.7 und 4.8. In den stark überbestimmten Szenarien hingegen, nimmt die Wahrscheinlichkeit eines Fehlschlags wieder ab und verbleibt im einstelligen Prozentbereich.

Eine Methode namens *FlipParticle* wurde implementiert. Sie sollte diese Fehlschläge verhindern, indem sie die Partikel versuchsweise auf die andere Seite einer Ebene spiegelt. Der Vorgang strapaziert die Stabilität der Simulation enorm, da mit der neuen, weit entfernten Position ein extrem abseitiger Zustand mit großem Fehler provoziert wird. In einigen Fällen konnte so jedoch aus einem lokalen Minimum herausgesprungen werden und ein Minimum gefunden werden, das ein *gültiges* Ergebnis liefert, siehe Abb. 4.9. Die Spiegelung eines Punktes an einer mehrdimensionalen Ebene besteht aus zwei Teilen: Zunächst wird die Ebene, die durch eine Menge von  $n$  Partikeln vorgegeben ist, durch multiple Regression bestimmt und der Normalenvektor berechnet, siehe Abschnitt 3.3.1f. Anschließend wird der Punkt unter Verwendung der Hesseschen Normalform gespiegelt.

Manchmal reichte es aus, einen Durchlauf erneut zu starten, wobei andere, zufällige Startwerte verwendet wurden. Es gibt jedoch Szenarien, für die trotz aller Anstrengungen kein *gültiges* Ergebnis gefunden werden konnte, obwohl ein solches existieren muss – schließlich wurden seine Positionen ohne Verwendung eines künstlichen Fehlers erzeugt.

Zwischenzeitlich wurde vermutet, dass die Verzerrung des Raumes ausschlaggebend sein könnte. Eine Einheit in  $t$ -Richtung entspricht  $c$  Einheiten in  $x/y/z$ -Richtung. Der Richtungsvektor  $\mathbf{N}_0$  kompensiert zwar die Stauchung in der  $t$ -Dimension, doch die Simulation selbst trägt bei der Integration und Dämpfung der linearen Verzerrung nicht Rechnung: Die Dämpfung wirkt quadratisch, somit könnte sie in der gestauchten  $t$ -Richtung unterproportional wirken. Testweise wurde die Schallgeschwindigkeit auf  $c = 1 \text{ m/s}$  gesetzt. Am Prinzip der Ortung ändert sich dadurch nichts, die Stauchung des Raumes ist aber aufgehoben. Die durchgeführten Experimente führten weder zu einer Erhöhung, noch zu einer Verminderung des Anteils fehlgeschlagener Szenarien. Auch die Raumverzerrung kann deshalb als Verursacher ausgeschlossen werden.

Das Problem der lokalen Minima scheint aufgrund der hohen Dimensionszahl (alle Freiheitsgrade) komplex zu sein. Vermutlich müssten testweise alle Objekte an allen



**Abbildung 4.9:** Wirkungsweise von FlipParticle in einer 2D-Lokalisierung. *Oben:* Position der Schallquellen. *Unten:* Quadratischer Fehler der Schallquelle in Bezug auf die momentanen Mikrofonpositionen. In Iterationsschritt 6541 wird die zufällig gewählte Schallquelle 2 an der durch die Mikrofone gegebenen Gerade gespiegelt. Die Spiegelung führt zur Auflösung des zuvor bestehenden lokalen Minimums.

Ebenen gespiegelt werden. Vielleicht könnte man Stat-Bat (vgl. Abschnitt 2.3) verwenden, um gute Initialwerte zu erhalten, und so eine bessere Voraussetzung für eine *gültige* Lokalisierung ohne lokale Minima zu schaffen.

## 4.8 Zusammenfassung

Mit dem Lampshade-Algorithmus konnte ein vergleichsweise schneller und leistungsstarker Algorithmus zur wechselseitigen Lokalisierung von Mikrofonen und Schallquellen geschaffen werden. Aufgrund seiner Flexibilität ist er für einen weiten Bereich von laufzeitbasierter Lokalisierung verwendbar, dies beinhaltet auch Standardfälle wie bekannte Mikrofonpositionen.

Mit dem Constraint-System ist er vorbereitet auf eine unvollkommene Zuordnung von Timestamps zu logischen Schallquellen, was im realen Experiment wegen Verdeckung von Empfängern (Mikrofon außerhalb der Signalreichweite) aktuell wird. Die Zuordnung kann beliebig vorgegeben werden.

Aus bislang ungeklärten Gründen landet der Algorithmus zuweilen in einem lokalen Minimum, obwohl ein globales, korrektes Minimum für die erzeugten Experimentalfälle existieren muss. Unklar ist, ob es sich um eine der wechselseitigen Lokalisierung immanente Schwierigkeit handelt, oder um eine Schwäche des Algorithmus. Mit einem durchgeführten Experiment wurde die Häufigkeit dieser lokalen Minima quantisiert. Desweiteren wurde die in Abschnitt 3.2 vorausgesagte, für eine eindeutige Lokalisierung benötigte Objektanzahl bestätigt.

In weiteren Experimenten ist eine Implementation des Newtonschen Iterationsverfahrens zur Lösung der wechselseitigen Lokalisierung zu untersuchen. Bei diesem handelt es sich um eine Standardmethode zur Lösung nichtlinearer Gleichungssysteme, wohingegen der proprietäre *Lampshade*-Algorithmus mit Herkunft aus der grafischen Simulation seine Existenzberechtigung erst beweisen muss. Zu erwarten ist eine Verbesserung der Laufzeit. Die Erfolge mit dem *Lampshade*-Algorithmus ließen jedoch zum Zeitpunkt dieser Masterarbeit die Entwicklung eines weiteren Algorithmus', dessen überlegene Leistung ungewiss ist, als nicht sinnvoll erscheinen. Der Fokus wurde statt dessen auf die Integration mit dem Timestamping gelegt, um eine reale Schallortung zu ermöglichen.

Da gezeigt werden konnte, dass der Algorithmus in der Theorie funktioniert, ist es nun hochgradig interessant, herauszufinden, ob er auch mit realen Werten zusammenarbeitet. Im zweiten, nun folgenden Teil dieser Arbeit wird eine Software *BasicTimestamping* entwickelt, die Schallsignale über die Mikrofone von Computern aufnimmt und Timestamps als Eingabewerte für die Lampshade-Lokalisierung generiert.

## 5 Timestamping

Im zweiten Teil dieser Arbeit wird eine Software entwickelt, die akustische Signale an handelsüblichen PCs aufzeichnen kann und markante Schallereignisse mit einem Zeitstempel markiert. *BasicTimestamping* wird in der Lage sein, über Netzwerk mit anderen Instanzen der Software zu kommunizieren und eine einheitliche zeitliche Referenz zu schaffen, eine Zeitsynchronisation. Auf diese Weise können die Zeitstempel zueinander in Relation gesetzt werden.

BasicTimestamping soll zum einen reales Datenmaterial liefern, um ein realistisches Modell zur Szenarienerzeugung für den Lampshade-Algorithmus zu finden. Es wird überprüft, welche Fehlerverteilung in welcher Größenordnung vorliegt. Zum anderen ist das Ziel, die Daten von BasicTimestamping als Eingabewerte für die Lokalisierung zu verwenden: Reale Schallquellen sollen geortet werden.

Die Funktionalität von BasicTimestamping beruht auf der Messung mehrerer fehlerbehafteter Messwerte, aus denen sich die Zeitstempel zusammensetzen. Sie müssen so genau ermittelt werden, wie irgendwie möglich, denn die Präzision der schlechtesten Messung bestimmt die Präzision des gesamten Vorgangs:

- Die Zeitsynchronisation zwischen den PCs bestimmt den zeitlichen Offset  $t_{\text{offset}}$  zwischen den Uhren.
- Über die Funktionen des Soundkartentreibers erfolgt ein Abgleich zwischen erwarteter und tatsächlicher Pufferposition.
- Die Verarbeitung der aufgezeichneten Samples muss korrekt und präzise erfolgen.

In all diesen Komponenten sind Glättungen der verrauschten Messreihen notwendig, die verschiedenen Anforderungen gerecht werden müssen. Hierfür stehen mehrere Verfahren zur Verfügung, etwa Mittelwert, exponentielle Glättung oder Regression. In einem Exkurs werden zunächst die Glättungsmethoden, die in BasicTimestamping zur Anwendung kommen, angeführt. Anschließend werden die Komponenten Netzwerkprotokoll, Zeitsynchronisierung und Signalverarbeitung erläutert.

## 5.1 Glättung von Messreihen

Während der Durchführung des Timestamping wird mehrmals eine verrauschte Messreihe  $X$  gemessen. Beispiele sind die Zeitsynchronisation zwischen Rechnern mit variabler Paketlaufzeit der Synchronisationssignale oder die Bestimmung der hardwareseitigen Aufnahmezeit bei der Signalaufzeichnung. Es existieren verschiedene Methoden, um das Rauschen zu eliminieren.

### 5.1.1 Arithmetischer Mittelwert

Ist der Messwert normal- oder gleichverteilt und ist von einem konstanten Erwartungswert auszugehen, so kann man den arithmetischen Mittelwert (Erwartungswert) über die einzelnen Messwerte  $X_i$  bilden mit

$$E(X) = \frac{1}{n} \sum_{i=1}^n X_i \quad (5.1)$$

Ein kontinuierlicher Strom von Daten wird nicht über alle Werte gemittelt, sondern nur über die  $n$  neuesten, wir erhalten einen *gleitenden Mittelwert*. Die Größe von  $n$  bestimmt die Stärke der Glättung.

Der Nachteil dieser Methode ist, dass ein ungünstiger Messwert, der weit vom Erwartungswert entfernt ist, solange mit konstantem Gewicht im Filter bleibt, bis er  $n$  Messungen später verworfen wird. Dabei ergibt sich ein Rücksprung der Mittelung, den man so spät nicht mehr erwarten würde. Weitere Nachteile sind die Berechnungskomplexität und der Speicherverbrauch, der abhängig von der Filtergröße ist.

### 5.1.2 Exponentielle Glättung

Eine Alternative ist ein Glättungsfiler unter Verwendung von *exponentieller Glättung* [4], bei der eine Gewichtung (Konvexkombination) zwischen altem Wert und neuem Messeinfluss vorgenommen wird. Der Glättungswert zum Zeitpunkt  $k$  berechnet sich rekursiv zu:

$$L_{n,k}(X) = \left(1 - \frac{2}{n}\right) L_{n,k-1} + \left(\frac{2}{n}\right) X_k \quad (5.2)$$

Mit  $\alpha = \frac{2}{n}$  im Koeffizienten erreicht die Glättungsfunktion bei gleichem  $n$  eine vergleichbare Wirkung wie der arithmetische Mittelwert. Die Verlaufskurve der exponentiellen Glättung ist bei Fehlmessungen vorteilhafter als der Mittelwert. Zwar schlägt sie beim Auftreten einer ungünstigen Messung stärker aus, sie regeneriert sich dafür sehr schnell wieder zu einem sinnvollen Wert hin, anstatt den Fehler über  $n$  Messungen zu tragen. Der zweite Vorteil ist die konstante Laufzeit.

Der Glättungsfilter  $L_n$  eignet sich gut zur Näherung konstanter Werte. Bleibt die Messung aber nicht konstant, sondern ändert sich linear mit konstanter Änderungsrate, so bleibt der Filter systematisch hinter der idealen Mittelung. Diese unbekannte Änderungsrate kann mittels *doppelter exponentieller Glättung* [14] bestimmt werden. Dazu werden die geglätteten Werte  $L_n$  nochmals geglättet, man erhält die Schätzung:

$$L'_{n,k}(X) = \left(1 - \frac{2}{n}\right) L'_{n,k-1} + \left(\frac{2}{n}\right) L_{n,k} \quad (5.3)$$

Den Prognosewert  $\hat{y}_{k+1}$  erhält man durch:

$$\hat{y}_{k+1} = 2L_{n,k} - L'_{n,k-1} \quad (5.4)$$

Ein Beispiel für eine lineare Messung mit konstanter Änderung ist die Zeitmessung an einer Uhr. Die Referenzzeit entspricht der x-Achse eines Schaubilds, die gemessene Uhr entspricht einer Geraden, die steigt oder fällt oder konstant ist, falls beide Uhren gleich schnell gehen. Die doppelte Glättung hätte hierfür verwendet werden können, jedoch verspricht die lineare Regression eine noch bessere Schätzung der Geraden.

### 5.1.3 Lineare Regression

Bessere Ergebnisse konnten erzielt werden, indem die letzten  $n$  Werte der Messreihe mit einer linearen Regression [3, 4] genähert wurden. Hierbei geht man von zwei Zufallsvariablen  $X, Y$  aus und vermutet linearen Zusammenhang und normalverteiltes Rauschen  $\epsilon$  mit  $E(\epsilon) = 0$ . Man erhält ein Modell  $Y = \alpha + \beta X + \epsilon$ . Ziel ist es, eine Gerade  $y = a + bx$  so durch die Messreihe  $X$  zu legen, dass der Fehler  $E$  minimiert wird. Ein häufig verwendetes Kriterium ist die Minimierung der Quadratsumme der Residuen, auch *Methode der kleinsten Quadrate* genannt. Wir definieren:

$$E = \min_{a,b \in \mathbb{R}} \sum_{i=1}^n (y_i - (a + bx_i))^2 \quad (5.5)$$

Nun berechnen wir den y-Achsenabschnitt  $a$  und die Steigung  $b$  der Geraden. Die folgenden Gleichungen erfüllen das Kriterium der Minimierung von  $E$ :

$$b = \frac{n \sum_{i=1}^n X_i Y_i - (\sum_{i=1}^n X_i) (\sum_{i=1}^n Y_i)}{n \sum_{i=1}^n X_i^2 - (\sum_{i=1}^n X_i)^2} \quad (5.6)$$

$$a = \frac{1}{n} \left( \sum_{i=1}^n Y_i - b \sum_{i=1}^n X_i \right) \quad (5.7)$$

Den Funktionswert der aktuellen Messung erhalten wir mit  $\tilde{y} = a + bX_n$ . Diese Methode führt zu einer besseren Mittelung linearer Messreihen als der Glättungsfilter. Sie schwingt nicht um das Optimum, und sie folgt der Messung nicht verzögert. Allerdings muss, wie beim arithmetischen Mittelwert, eine Laufzeit in  $O(n)$  zur Aufsummierung

der Messwerte in Kauf genommen werden. Die Implementierung entstammt einer Vorlage von David Swaim [15]. Dieser einfache Fall der Regression kommt ohne Berechnung einer inversen Matrix aus und ist deshalb ohne anspruchsvolle Algebrabibliotheken einsetzbar. Auch hier ist doppelte Fließkommapräzision erforderlich.

## 5.2 Netzwerkkommunikation

Für das Timestamping ist eine drahtlose oder drahtgebundene Netzwerkverbindung der Geräte untereinander erforderlich. Zum einen erfolgt die Zeitsynchronisation der Clients über ein IP-Netzwerk, zum anderen müssen für die „Online“-Lokalisierung die Zeitstempel ausgetauscht werden.

Grundlegendes Prinzip des rudimentären Peer-To-Peer-Netzwerks ist die Aushandlung eines *Master*-Hosts, während die anderen Rechner *Clients* sind. Der Master wird als Referenz für die Uhrzeit und die Uhrengeschwindigkeit festgelegt, an welcher sich alle anderen Rechner orientieren. Sind alle Uhren mit dem Master synchronisiert, so sind sie auch untereinander synchronisiert.

Wird die Software auf einem Client mit LAN-Verbindung (Local Area Network) gestartet, so beginnt er, seine korrekte Priorität in der Liste der Clients auszuhandeln. Dafür wurde ein einfaches Protokoll entworfen.

Zunächst vergibt der Client sich selbst eine Prioritäts-Zahl „ID“ mit der niedrigsten Priorität HIGH\_ID. Nun werden drei Fälle unterschieden:

- Findet er keine anderen Rechner, so setzt er  $ID = 0$  und sich dadurch zum Master, da er die niedrigste ID hat.
- Findet er andere Clients, so sucht er zunächst nach der niedrigsten ID und akzeptiert diesen Host als Master. Danach sucht er nach der höchsten ID und setzt seine eigene ID auf die nächstgrößere, freie ID.
- Trägt ein Rechner dieselbe ID wie ein anderer, so erhöhen beide ihre ID um eine Zufallszahl  $0 < x < S$  ( $S$  ist die Schrittweite zwischen zwei IDs) und scannen das Netzwerk erneut, bis sie eine eindeutige Nummer besitzen. Dieser Fall kann auftreten, wenn sich zwei Rechner gleichzeitig einloggen. Beide tragen noch die HIGH\_ID, was als „nicht vorhanden“ zählt und geben sich dieselbe ID.

Geht ein Client-Rechner offline, so hat das keine Auswirkungen auf das Netzwerk. Loggt sich hingegen der Master aus, so übernimmt der Rechner mit der nächsthöheren ID seine Rolle. Dessen eigene, bislang aktive Synchronisation wird zurückgesetzt und deaktiviert.

Die Liste der Rechner erlangt ein Client, indem er per UDP ein `HELLO`-Paket verbunden mit seiner ID an die UDP-Broadcast-Adresse des LAN schickt. Dieses wird von allen Clients empfangen und mit einem `WELCOME`, verknüpft mit der ID, beantwortet.

### 5.3 Zeitsynchronisation

Im algorithmischen Teil der Lokalisierung in Kapitel 4 wurde stillschweigend davon ausgegangen, dass die Uhren aller Mikrofone identisch sind und stets dieselbe Zeit anzeigen. Bei einem allgemeinen Versuchsaufbau ist jedoch davon auszugehen, dass die Mikrofone an verschiedenen Geräten angeschlossen sind, in unserem Fall brauchen wir  $m$  Laptops für  $m$  Mikrofone. Jedes dieser Geräte besitzt eine eigene Uhr. Die Lokalisierung benötigt eine Synchronisation dieser Uhren, damit sie die relativen Zeitstempel miteinander vergleichen kann. Fehler bei der Synchronisation schlagen sich in direkter Weise auf den gesamten Fehler der Lokalisierung nieder.

Beispiel: Die Uhren zweier Geräte laufen identisch schnell, aber mit einer Abweichung von 1 ms. Die zwei Geräte empfangen zur selben Zeit ein Signal. Die Uhren melden nun zwei Timestamps mit einer relativen Abweichung von 1 ms an die Lokalisierung, welche die Ortung dementsprechend berechnet.

#### 5.3.1 NTP-Protokoll

Das *Network Time Protocol* (NTP) ist ein Verfahren zur Zeitsynchronisation zwischen Computersystemen. Ursprünglich war geplant gewesen, dieses Protokoll zur Schaffung einer gemeinsamen Zeitbasis in *BasicTimestamping* zu verwenden, denn es ist weit verbreitet und gilt als leistungsfähig. Über das öffentliche Internet ist es in der Lage, eine Genauigkeit von ca. 10 ms einzuhalten, über kabelgebundene lokale Netzwerke und unter idealen Bedingungen sogar bis zu 0,2 ms und weniger [4].

Leider konnten fertige NTP-Clients lediglich die *Real Time Clock* (RTC) des PC synchronisieren. Diese besitzt nur eine Sekunden-Auflösung und geht so ungenau, dass im Versuch bereits nach ein paar Sekunden wieder ein Zeitunterschied von mehreren Millisekunden auftrat. Der Schwerpunkt der gängigen NTP-Projekte lag eher bei der ungefähren Angleichung der Systemuhren von Computern auf dieselbe Sekunde, ein Ziel, welches der Windows-eigene NTP-Client `W32Time` nicht einmal erreicht, zumindest nicht unter Windows Server 2003 [16]:

The W32Time service cannot reliably maintain sync time to the range of 1 to 2 seconds. Such tolerances are outside the design specification of the W32Time service.

Darüber hinaus gibt es das von der Zürich University of Applied Sciences in Winterthur entwickelte *Precision Time Protocol* (PTP), welches den Schwerpunkt auf Präzision im lokalen Netz setzt [17]. Die Informationen auf der Website erschienen vielversprechend. Die Einarbeitung in die umfangreiche PTP-Software und ihre Adaption zeichnete sich jedoch als kompliziert ab. Deshalb wurde beschlossen, die Zeitsynchronisation eigens zu entwickeln, was nicht aufwändiger erschien, als die bisherige Recherche und die erfolglosen Versuche mit NTP oder PTP.

### 5.3.2 Grundlagen

Das allgemeine Prinzip der Synchronisation ist, dass der Client einen Offset berechnet, der den Unterschied seiner Uhr  $T_C$  zur Uhr des anderen Computers  $T_H$ , des *Hosts*, angibt. Nachbedingung der Synchronisation soll sein:

$$T_C = T_H - t_{\text{offset}} \quad (5.8)$$

Bei der Synchronisation sind zwei Fehlertypen zu unterscheiden: Der *systematische* Fehler ist eine konstante Abweichung zweier Uhren. Er tritt beispielsweise auf, wenn versäumt wurde, die Netzwerklatenz eines stabilen Netzwerks (d.h. der Betrag der Paketlaufzeit ist annähernd konstant) einzukalkulieren. Ansonsten lässt er sich bei der Zeitsynchronisation vollständig eliminieren und ist nicht von Bedeutung. Maßgeblich ist der *statistische* Fehler. Er resultiert aus der Abfragegenauigkeit der Uhren, und aus der unterschiedlichen Laufzeit der Datenpakete beim Austausch der Uhrzeit. Die Qualität der Synchronisation lässt sich durch die Größe des zufälligen Fehlers ausdrücken. Im Allgemeinen ist sie von der Größe und Streuung der *Round Trip Time* (RTT) abhängig. Das Hauptaugenmerk lag auf der Reduzierung der Streuung beim Abgleich der Systemuhren.

### 5.3.3 Implementierung

Das Basiselement der Synchronisation ist der *Ping*: Will der Client sich mit einem Host synchronisieren, so schickt er ein IP-Paket an den Host-Rechner und merkt sich die Uhrzeit des Versandes. Der Host antwortet so schnell wie möglich mit dem Zeitstempel seiner Uhr  $T_H$ . Sobald der Client die Antwort empfangen hat, misst er die seit dem Versand vergangene Zeit  $t_P$ , die *Round Trip Time*. Desweiteren wird die eigene Zeit  $T_C$  zum Zeitpunkt der Ankunft festgehalten. Nun berechnet man den Offset  $t_{\text{offset}}$ :

$$t_{\text{offset}} = T_H - T_C + \frac{1}{2}t_P \quad (5.9)$$

Zwischen dem Zeitpunkt, zu dem der Host seinen Zeitstempel  $T_H$  verschickt hat und der Differenzbildung zum Zeitpunkt  $T_C$  ist eine unbekannte Zeitdauer verstrichen, in

der das Paket im Netzwerk unterwegs war. Deshalb wird eine Verzögerung hinzuaddiert. Insgesamt sind  $t_P$  Zeiteinheiten vergangen. Geht man davon aus, dass das Paket in beide Richtungen gleich lang unterwegs war, so benötigte es für die Rückstrecke  $\frac{1}{2}t_P$ . Wird der Client nach einem aktuellen Zeitstempel gefragt, so gibt er den synchronisierten Zeitstempel  $S(T_C) = T_C + t_{\text{offset}}$  zurück, welcher der Uhrzeit auf dem Host  $T_H$  entspricht.

Diese rudimentäre Synchronisierung wäre anfällig für Schwankungen der Paketlaufzeit. Deshalb wird pro Synchronisations-Vorgang nicht nur ein Ping verschickt, sondern mehrere. Danach verarbeitet ein zweistufiger Prozess alle eingehenden Pakete.

### 5.3.4 Vorverarbeitung

Zunächst wird davon ausgegangen, dass ungewöhnlich hohe Ping-Zeiten für unbrauchbare Pakete sprechen und diese auszusortieren sind. Was als „hoch“ einzustufen ist, kann nicht mit konstanten Parametern bestimmt werden. Die Synchronisation soll sowohl im lokalen Netz mit Pingzeiten um 1 ms funktionieren, als auch im Internet, wo ein Ping von 20 ms ein typischer Wert ist. Eine Methode ist, die Werte  $X_i$  der Messreihe  $X$  anhand ihrer Abweichung vom Mittelwert  $E(X)$  zu klassifizieren. Ihre Standardabweichung ist  $\sigma_X$ . Weicht ein Messwert um mehr als  $2\sigma_X$  vom Mittelwert ab, so gilt er als ungewöhnlich und wird entfernt. Genauso fallen besonders niedrige Werte weg, die auf eine fehlerhafte Messung hinweisen. Im Intervall  $E(X) \pm 2\sigma_X$  einer normalverteilten Messung liegen ca. 95 % aller Messpunkte, es werden also ca. 5 % der Werte verworfen. Die verbleibenden Messwerte bezeichnen wir mit  $\tilde{X}$ .

### 5.3.5 Drift und Regression

Durch die letzten  $n$  dieser vorselektierten Werte  $\tilde{X}$  wird mittels linearer Regression eine Gerade gelegt. Die Regressionsfunktion liefert den  $y$ -Achsenabschnitt  $a$  und die Steigung  $b$  der Geraden zurück.

Wären beide Uhren ideal, das heißt, liefen sie mit identischer Geschwindigkeit, so wäre  $b = 0$ . In der Realität besitzen die verwendeten *High Precision Event Timer* (HPET) eine zur Referenzzeit relative Gangungenauigkeit, einen *Drift*. Er besitzt die Einheit s/s, was sich auch als prozentualer Wert angeben lässt.

Man kann davon ausgehen, dass die Uhren beider Rechner zwar unterschiedlich schnell, aber mit konstanter Geschwindigkeit gehen und Messunterschiede ausschließlich auf Messfehler zurückzuführen sind. Das Experiment bestätigte diese Annahme. Das bedeutet, wir können eine lineare Regression anwenden. Im Gegensatz zu der zunächst verwendeten einfachen exponentiellen Glättung, die den Drift vernachlässigte, schlägt

die lineare Regression „zwei Fliegen mit einer Klappe“ und ermittelt sowohl den Offset als auch die Driftrate. Wir erhalten:

$$\tilde{t}_{\text{offset}} = a + T_C \cdot b \quad (5.10)$$

$$\tilde{t}_{\text{drift}} = b \quad (5.11)$$

Wir vervollständigen die Synchronisation und interpolieren die Zeit zwischen zwei Synchronisationsvorgängen.  $T_{\text{old}}$  ist der Zeitpunkt der letzten Synchronisation.

$$S(T_C) = T_C + \tilde{t}_{\text{offset}} + \tilde{t}_{\text{drift}} \cdot (T_C - T_{\text{old}}) \quad (5.12)$$

Im Experiment mit einem Desktop-Rechner und einem Laptop der 2-GHz-Klasse hat sich gezeigt, dass der Drift nicht zu vernachlässigen ist. Sogar die modernen HPET-Uhren zeigten eine relative Abweichung von ca. 2 ms bei einem Synchronisationsintervall von 10 s, was 0,02 % entspricht. Zwar würde der Fehler bei der nächsten Synchronisation wieder getilgt, doch im Mittelwert hätte man stets eine Ungenauigkeit von einer Millisekunde, die man nur durch kürzere Synchronisationsintervalle verringern kann. Für das geplante Vorhaben war das ein unnötig großer Fehler, deshalb ist die Driftkorrektur eine wichtige Verbesserung.

Experimentell wurde der verbleibende Fehler ermittelt. Zunächst war der Ping des mit WLAN ans Netzwerk angeschlossenen Rechners ein großes Problem. Im näheren Umfeld des Versuchsaufbaus scheinen sich eine Vielzahl von Störern wie Funktelefone oder benachbarte WLANs zu befinden. Die Pingdauer lag meistens bei ca. 1 ms, zeigte aber je nach Tageszeit eine Häufung bei 5 ms und viele Ausreißer auf mehr als 40 ms. Bevor die Glättung implementiert war, wurde dadurch die Synchronisation trotz Vorverarbeitung der Messwerte massiv gestört. Nach Einführung der Regression hingegen konnte das Offset-Update stabilisiert werden. Die Genauigkeit der Synchronisation konnte gemessen werden, indem vor einer Synchronisation der zu erwartende Offset-Wert vorhergesagt wird. Das ist zwar nur eine rudimentäre Überprüfung, weil die geglättete Vorhersage exakter ist als der eintreffende Messwert, trotzdem lassen sich Tendenzen ausmachen.

Die Präzision liegt nun bei weniger als 10 % der Round Trip Time. Liegt sie stabil bei 1 ms, so ist eine Genauigkeit von ca. 0,01 ms erreichbar. Beim angesprochenen, instabilen Netzwerk mit variierender RTT verschlechtert sich die Synchronisation auf 1 ms und darüber. Ähnliche Werte erreicht auch das Verfahren *NTP*.

## 5.4 Signalverarbeitung

Den Kern des Timestamping stellt die Aufnahme und Auswertung des Schallsignals dar. Über das an eine Soundkarte angeschlossene interne oder externe Mikrofon wird

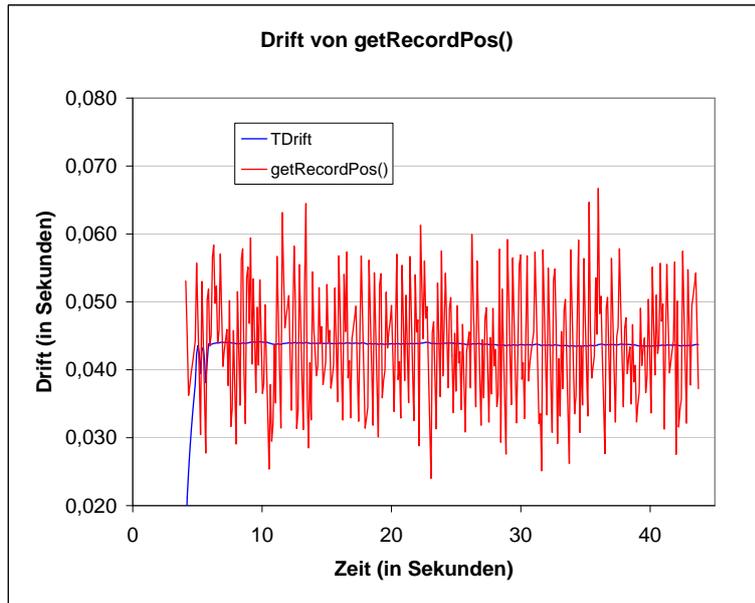
das ankommende Signal zunächst kontinuierlich aufgezeichnet und in einen Ringpuffer von mehreren Sekunden Dauer gespeichert. Anschließend wird auf dem aufgezeichneten Signal ein Algorithmus angewendet, der Schallereignisse detektiert, ein charakteristisches Merkmal darin sucht und diesem einen exakten Zeitpunkt, einen *Timestamp*, zuordnet.

Die Schnittstelle zur Hardware wird durch die Sound-Bibliothek *FMOD* [18] hergestellt. Sie kommt in vielen bekannten Softwareprojekten zum Einsatz und wurde auch schon in der Billardsimulation *HapticBillard* [12] zur positionalen 3D-Wiedergabe von Geräuschen verwendet.

Aufnehmen von Sound mit FMOD ist vergleichsweise unkompliziert. Der zur Aufnahme verwendete zirkuläre Audio-Puffer wird von FMOD selbst erstellt. Danach startet man die Aufnahme mit `recordStart()` und FMOD zeichnet in diesen Puffer kontinuierlich die von der Soundkarte gelesenen Daten auf. Ein sinnvoller Wert für die Puffergröße sind 10 Sekunden. Das heißt, dass vergangene Schallsignale nach einem Durchlauf von 10 Sekunden verworfen werden. Ihre Auswertung muss zuvor erfolgt sein – anschließend werden sie nicht mehr benötigt. Auf diese Weise lässt sich die notwendige Speicherkapazität begrenzen, anstatt dass ein Puffer für die komplette Aufnahmedauer vorgehalten werden muss.

Das aufgenommene Signal wird nun verarbeitet. FMOD stellt hierfür die `lock()`-Funktion zur Verfügung, die einen beliebigen Teil des Puffers für FMOD sperrt und einen Pointer auf diesen Speicherbereich zurückgibt. Es dürfen nur Bereiche gesperrt werden, in die FMOD nicht zu schreiben versucht. Der zu verarbeitende Bereich liegt zwischen einer Indexmarke  $I_m$  und der Funktion `getRecordPosition()`, hier bezeichnet als Index  $I_{\text{pos}}$ , welcher die Position angibt, an der FMOD gerade aufnimmt. Zunächst ist  $I_m = 0$ . Nun wird die Analyse im Intervall  $[I_m, I_{\text{pos}}]$  durchgeführt. Erfolgt ein Umbruch des Puffers, so kann man das daran erkennen, dass  $I_{\text{pos}}$  kleiner ist als  $I_m$ . In diesem Fall erfolgt eine getrennte Analyse von  $I_m$  bis zum Ende des Puffers und dann von  $I_0$  bis  $I_{\text{pos}}$ . Nach der Analyse wird  $I_m$  auf  $I_{\text{pos}}$  gesetzt und der Puffer mit `unlock()` wieder freigegeben. Auf diese Weise wird sichergestellt, dass jedes aufgenommene Sample genau ein mal verarbeitet wird, also keines übergangen und keines doppelt analysiert wird. Wie oft die Auswertung erfolgt, d.h. wie groß der zeitliche Abstand zwischen zwei Auswertungen ist, spielt für das Ergebnis keine Rolle. Es hat sich als sinnvoll erwiesen, die Auswertung alle 100 ms aufzurufen.

Eine Aufnahme in 16-bit Stereo mit  $f_{\text{rec}} = 44.100$  Hz benötigt pro Sekunde ca. 176 KB Speicher. Der Puffer benötigt also ca. 1,76 MB Speicher. Wird die Auswertung alle 100 ms aufgerufen, so könnte der Puffer problemlos auf 1 Sekunde und weniger verkleinert werden. Auf die Stereoaufnahme könnte verzichtet werden, da viele Mikrofone nur einen Kanal unterstützen und die Stereokanäle während der Analyse zu einem Signal gemixt werden. Für mobile Geräte kann der Speicherverbrauch für den Puffer mit unter 100 KB also klein gehalten werden.



**Abbildung 5.1:** `getRecordPosition()` (rot) rauscht mit einer Standardabweichung von  $\sigma = 8,5$  ms. Die Mittelung (blau) pendelt sich dagegen präzise bei 44 ms ein. Die Abfrage erfolgt alle 100 ms, die Samplingfrequenz ist 44.100 Hz.

Die Auswertung folgt der Aufnahme in variierendem zeitlichen Abstand. Bei der Abfrage von `getRecordPosition()` wird der exakte, synchronisierte Zeitstempel  $T_{\text{pos}} = S(T_C)$  für die zurückgegebene Sample-Nummer  $I_{\text{pos}}$  gespeichert. Dadurch ist bekannt, wann der Recorder an dieser Position aufgenommen hat. Wurde ein Timestamp bei einem beliebigen, zurückliegenden Sample  $I_{\text{sample}}$  detektiert, so kann dessen Zeitpunkt  $T_{\text{sample}}$  mittels der dazwischen liegenden Zeit berechnet werden:

$$T_{\text{sample}} = T_{\text{pos}} - \frac{I_{\text{pos}} - I_{\text{sample}}}{f_{\text{rec}}} \quad (5.13)$$

Leider rauscht die Funktion `getRecordPosition()` sehr stark. Sie gibt die zu erwartende Position mit einem statistischen Fehler von fast 10 ms zurück. Eine exakte Lokalisierung wäre so nicht machbar, siehe Abb. 5.1. Der Grund für die Ungenauigkeit der Funktion ist, dass das Positionsupdate vom Soundkartentreiber nicht Sample für Sample erfolgt, sondern blockweise in 512-Samples-Blöcken. Selbst das wäre unproblematisch, wenn das Update zu festen Zeitpunkten mit derselben Schrittweite geschehen würde. In dem Fall könnte man den Zeitpunkt des Updates exakt abpassen und zwischen zwei Updates interpolieren. Tatsächlich ist es aber so, dass die Anzahl der Blöcke, um die der Zeiger weiter wandert, scheinbar willkürlich zwischen 1 und 4 variiert. Die tatsächliche Position lässt sich so nicht feststellen, nur mitteln. Die Korrespondenz mit den Machern von *FMOD* ergab keine Lösung des Problems.

Eine Alternative wäre, sich den Startzeitpunkt der Aufnahme  $T_{\text{start}}$  zu merken und die gesamte Sample-Nummer, geteilt durch die Frequenz, zu addieren. Sie ergibt sich aus der aktuellen Position im Puffer mal der Anzahl der Pufferdurchläufe:

$$T_{\text{sample}} = T_{\text{start}} + c_{\text{wrap}} \cdot T_{\text{length}} + \frac{I_{\text{sample}}}{f_{\text{rec}}} \quad (5.14)$$

Leider funktioniert auch das nicht, da sich der Startzeitpunkt nicht exakt bestimmen lässt. Gibt man das Kommando `recordStart()`, welches die Aufnahme beginnt, und speichert den Zeitpunkt  $T_{\text{start}}$ , so stellt man beim Nachmessen eine konstante Abweichung im Bereich von  $\pm 100$  ms fest. Offenbar dauert es eine lange Zeit, bis der Soundtreiber alle Vorbereitungen zur Aufnahme getroffen hat. Auf diese Weise erhält man Zeitstempel mit minimalem, statistischem Fehler, aber einer massiven, systematischen Abweichung.

### 5.4.1 Geglätteter Drift

Es konnte eine Methode gefunden werden, die den zufälligen Fehler minimiert und trotzdem eine systematische Abweichung gänzlich verhindert. Indem die Werte  $I_{\text{pos}}$  massiv gemittelt werden, erhält man mit  $\tilde{T}_{\text{pos}}$  eine stabile Angabe, um wieviel die reale Aufnahmeposition von der berechneten Soll-Position abweicht:

$$\tilde{T}_{\text{pos}} = L_{400} \left( T_C - \frac{I_{\text{pos}}}{f_{\text{rec}}} - T_{\text{start}} - c_{\text{wrap}} \cdot T_{\text{length}} \right) \quad (5.15)$$

Das massive Glätten ist erlaubt, denn man kann davon ausgehen, dass der Timer  $T_C$  und die Aufnahme  $I_{\text{pos}}/f_{\text{rec}}$  mit derselben Geschwindigkeit laufen und daher um einen konstanten Zeitbetrag voneinander abweichen. Wir erhalten den vorläufigen Timestamp eines Samples:

$$T_{\text{sample}} = S(T_C) - T_C + \tilde{T}_{\text{pos}} + T_{\text{start}} + c_{\text{wrap}} \cdot T_{\text{length}} + \frac{I_{\text{sample}}}{f_{\text{rec}}} \quad (5.16)$$

$$= S(0) + \tilde{T}_{\text{pos}} + T_{\text{start}} + c_{\text{wrap}} \cdot T_{\text{length}} + \frac{I_{\text{sample}}}{f_{\text{rec}}} \quad (5.17)$$

Anstatt die synchronisierte Zeit  $S(T_C)$  abzufragen und anschließend wieder zu subtrahieren, setzen wir nur den driftkorrigierten Offset  $S(0)$  ein. Zuletzt ergänzen wir den manuell zu ermittelnden Korrekturwert  $T_{\text{latency}}$  zur Kompensation der Hardware-Latenz (Abschnitt 5.6). Damit erhalten wir die endgültige Formel für den Zeitpunkt eines Samples in der Aufnahme:

$$T_{\text{sample}} = S(0) + \tilde{T}_{\text{pos}} + T_{\text{start}} + c_{\text{wrap}} \cdot T_{\text{length}} + \frac{I_{\text{sample}}}{f_{\text{rec}}} - T_{\text{latency}} \quad (5.18)$$

Der in der Synchronisation angesprochene Drift der Timer ist nicht von Bedeutung, da zwischen Aufnahme und Auswertung maximal 100 ms vergehen. In dieser Zeit driftet der Timer nur um etwa 0,02 ms.

Ein weiteres, seltsames Phänomen trat auf. Bei einer bestimmten, ungeeignet gewählten Samplingrate driftete die Soundkartenposition in der Größenordnung von 100 ms pro 10 s, die Soundkarte nahm also fast 1 % zu langsam / zu schnell auf. Der Effekt hätte zu Problemen geführt, schließlich ändert sich die reale Zeitdauer  $T_{\text{length}}$  des Puffers und führt so in obiger Formel 5.15 zu Rechenfehlern. Der Fehler trat beim verwendeten Desktop bei 48.000 Hz auf und beim Laptop bei 44.100 Hz. Er war hingegen vermeidbar, indem man beim Desktop eine Samplingfrequenz von 44.100 Hz und beim Laptop von 48.000 Hz konfigurierte. Bei Verwendung des ASIO-Treibers *ASIO4ALL* [19] tritt der Fehler bei allen Rechnern bei 48.000 Hz auf, nicht aber bei 44.100 Hz. Eine funktionierende Samplingfrequenz muss also manuell gewählt werden. Mit solchen Phänomenen ist bei der Verwendung von Windows<sup>®</sup> als Betriebssystem vermutlich zu rechnen.

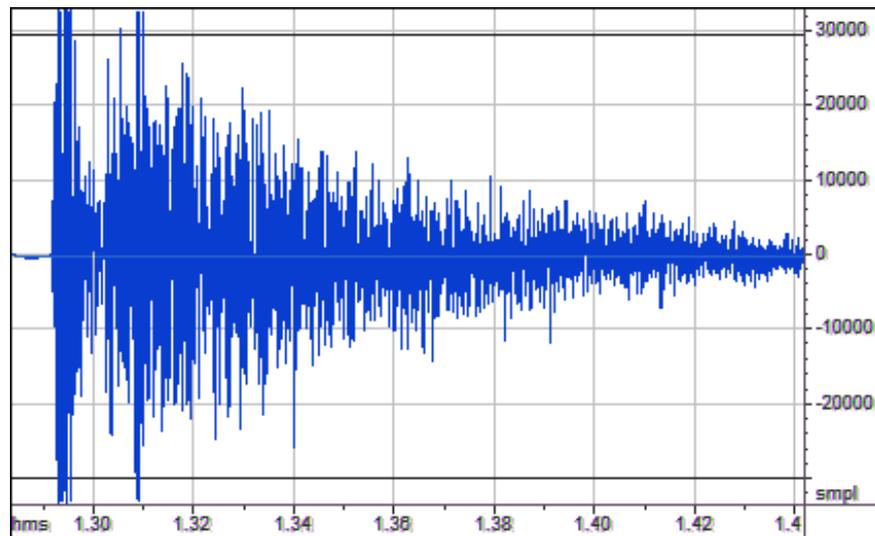
Nachdem eine Methode geschaffen war, einem Sample einen exakten Zeitpunkt zuzuordnen, musste nun untersucht werden, wie man die aufgezeichneten Signale weiterverarbeiten kann. Zunächst war es notwendig, eine Klassifikation zu schaffen, die Soundsignale erkennt und vom Hintergrundrauschen trennt. Danach musste aus diesen erkannten Soundsignalen ein Charakteristikum extrahiert werden, welches bei allen aufzeichnenden Geräten innerhalb des Signals möglichst identisch ist.

### 5.4.2 Schwellwertfunktion

Zur Erkennung von Klangereignissen wird eine Schwellwertfunktion verwendet. Das Signal wird bis zur aktuellen Position  $k$  durchlaufen und jedes Sample wird von Stereo nach Mono konvertiert und in einem zirkulären Puffer gespeichert. Aus dem Puffer wird der quadratische Mittelwert, der *RMS*-Wert der letzten  $n$  Samples, berechnet.

$$P_{\text{RMS}} = \sqrt{\frac{1}{n} \sum_{i=k-n}^{k-1} (S_i)^2} \quad (5.19)$$

Für die Puffergröße wird  $n = 128$  verwendet. Mit dieser Mittelung gibt der RMS-Wert die momentane Schall-Leistung für ein Signal ab 400 Hz gut wieder (ca. die Dauer einer Schwingung). Ein Tiefpassfilter mittelt diesen Wert über ca. 1 Sekunde und man erhält die durchschnittliche Schall-Leistung  $P_{\text{lowpass}} = L_{44100}(P_{\text{RMS}})$ . Übersteigt  $P_{\text{RMS}}$  die Schwelle  $k \cdot P_{\text{lowpass}}$  und war der Status zuvor „kein Signal“, so wird eine Marke „Signalbeginn“ gesetzt und der Status wechselt zu „Signal“. Sobald der RMS-Wert danach die Schwelle  $P_{\text{lowpass}}$  für die Dauer von  $w$  Samples unterschreitet während „Signal“ gesetzt war, wird der Status wieder zurückgesetzt zu „kein Signal“ und die Marke „Signalende“ wird gesetzt.



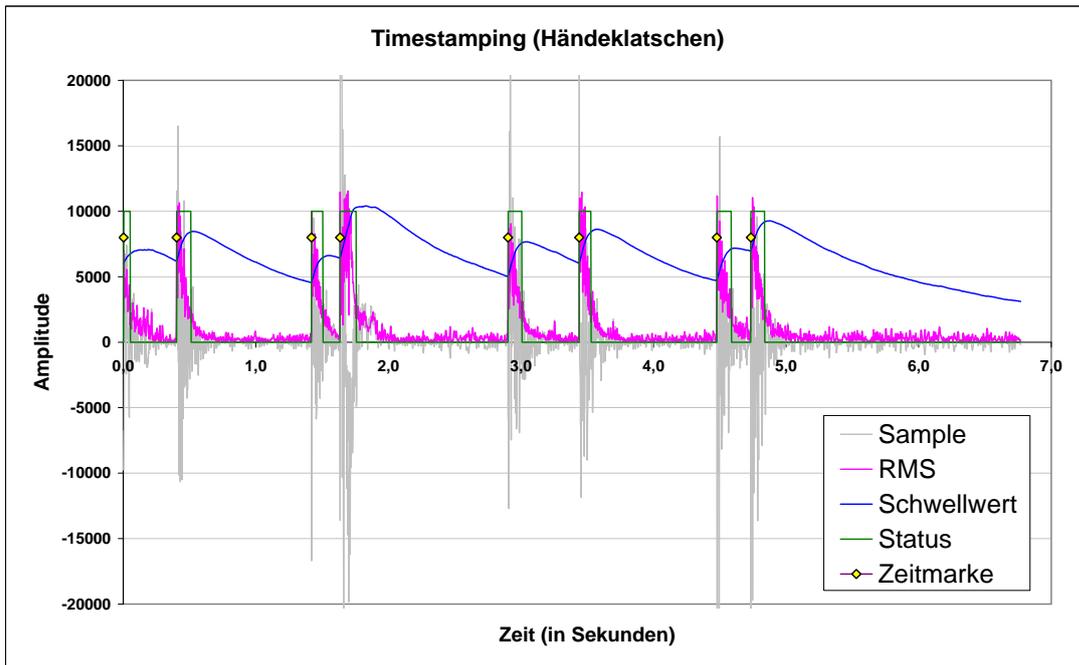
**Abbildung 5.2:** Ein Klatschen in die Hände dauert mehr als 100 ms und zeigt zwei Maxima, die 15 ms voneinander entfernt sind. Ein eindeutiges Maximum kann so nicht bestimmt werden.

Während ein Signal aufgenommen wird, steigt gleichzeitig der Tiefpassfilter an. Dies hat zwei vorteilhafte Folgen. Zum einen ist ein so detektiertes Signal nur von kurzer Dauer, denn bei gleichbleibender Leistung wird  $P_{lowpass}$  irgendwann so groß wie das Signal, so dass der Status zurückgesetzt wird. Zum anderen passt sich die Schwelle den Umgebungsgeräuschen an. Ist es im Hintergrund sehr leise, so sinkt der Schwellwert bis zur Rauschgrenze der Hardware ab und detektiert so auch sehr leise Signale. Ist es im Hintergrund dagegen vergleichsweise laut, so werden nicht alle eintreffenden Geräusche für Schallsignale gehalten, eine sinnvolle Wahl von  $k \in [4, \dots, 8]$  vorausgesetzt.

### 5.4.3 Charakteristisches Signalmerkmal

Nun sind der Anfang und das Ende eines Signals bekannt. Es muss nun möglichst exakt der Zeitpunkt dieses Signals bestimmt werden. Zunächst wurde versucht, den Zeitpunkt der maximalen Lautstärke des Signals zu finden. Das kann entweder durch direktes Beobachten des Samplingbetrages geschehen, oder durch RMS-Mittelung. Im Experiment zeigte sich jedoch, dass die so erzielten Ergebnisse sehr fehleranfällig waren. Am Beispiel vom „Händeklatschen“ wird das deutlich.

Klatschen, aber auch sprachähnliche Schallsignale, zeigen eine zufällig aussehende Verteilung der Samples bis zu einer Frequenz von 20 kHz. Es hört sich für den Menschen kurz an, in Wirklichkeit dauert es aber bis zu 100 ms. Vermutlich spielen hier Reflexionen von den Wänden eine Rolle. In den ersten 10 ms kann dabei der Dynamikumfang



**Abbildung 5.3:** Timestamping von mehrmaligem Klatschen in die Hände. Steigt der RMS-Wert (violett) über den Schwellwert (blau), so wird ein Signaltrigger (grün) geschaltet. Die resultierenden Zeitmarken (gelbe Karos) stehen auf dem Beginn des Signals.

von Mikrofon und Soundkarte mehrmals erreicht werden, das Signal übersteuert (Abbildung 5.2). Selbst mit RMS-Mittelung konnte auf diese Weise kein eindeutiges Signal ermittelt werden, falls es im Signal mehrere Peaks gab. Wählten zwei Geräte beim Aufzeichnen zwei verschiedene Peaks, etwa wie in der Abbildung, so entsteht ein massiver Fehler, obwohl die Bestimmung des einzelnen Peaks genau ist.

Diesem Problem wurde zunächst mit einer einfachen Variante begegnet. Offenbar ist der Zeitpunkt, zu welchem das Signal  $P_{\text{RMS}}$  den Schwellwert  $k \cdot P_{\text{lowpass}}$  übersteigt, nicht nur eindeutig, sondern bei „harten“ Signalen, wie zum Beispiel dem Schlagen eines Löffels gegen eine Glasflasche, auch relativ genau. Die dadurch erreichbare Genauigkeit liegt bei einem Bruchteil einer Millisekunde.

Weichere Geräusche, wie Sprache oder Klatschen, führen zu einer Verschlechterung der Genauigkeit bei der Schwellanstiegs-Bestimmung. Möglicherweise könnten die Peaks solcher Signale mit einer RMS-Maximumsuche präziser berechnet werden, doch die Ungenauigkeit des Schwellanstiegs liefert immer noch bessere Ergebnisse, als wenn zwei Geräte unterschiedliche Peaks eines Signals zum Signalmaximum erklären, der Schwellanstieg ist also robuster. Abbildung 5.3 zeigt einen typischen Schwellwertverlauf.

## 5.5 Experiment

Ein wesentlicher Bestandteil der Entwicklung der Software war die vielfach wiederholte Überprüfung ihrer Leistungsfähigkeit. Mit dem experimentellen Feedback wurden ihre Komponenten sukzessive verbessert. Nachdem die Software netzwerkfähig geworden war und eine einheitliche Zeitbasis besaß, wurden zwei Rechner miteinander verbunden und untersucht, um wieviel sich ein aufgenommener Zeitstempel auf beiden Rechnern unterschied. Das Ziel war, den Unterschied bei identischer Entfernung von einer Schallquelle auf Null zu reduzieren. Die zwei Fehlertypen des *systematischen* und des *statistischen* Fehlers wurden hierbei getrennt behandelt, da ihre Ursachen verschieden waren.

Mit der Vermeidung einer Fehlerquelle wurden jeweils neue, kleinere Fehlerquellen entdeckt. Der durch die Soundkartenlatenz entstehende Fehler von  $\pm 3$  ms zum Beispiel, war erst zu beobachten, nachdem das statistische Rauschen der Soundkartenposition von  $\pm 20$  ms kompensiert worden war.

Die Computer, die zum Einsatz kamen, waren ein Desktop-Rechner Athlon XP 2600+ mit Onboard Soundchip und ein Laptop Fujitsu Siemens Amilo Pro 3515 mit Core Duo-Prozessor. Als Betriebssystem kam in beiden Fällen Microsoft Windows<sup>®</sup> XP SP2 zum Einsatz. Die Netzwerkanbindung des Desktop-Rechners erfolgte durch LAN an einen DSL-Router Speedport W 500V, der Laptop war mit WLAN nach 802.11g angebunden. Die Funkanbindung geschah bewusst, da das Verhalten der Zeitsynchronisation in einem nicht idealen Netz untersucht werden sollte. Die CPU des Laptop unterstützt das *SpeedStep*-Verfahren zur dynamischen Anpassung der Taktfrequenz. Dieses beeinträchtigt die Soundkartenaufzeichnung und wurde daher deaktiviert und die CPU auf maximale Leistung eingestellt. Auch der Anbieter des kostenlosen ASIO-Treibers *ASIO4ALL* [19] empfiehlt im *Instruction Manual* diese Vorgehensweise.

### 5.5.1 Präzision

Auf die Präzision von *BasicTimestamping* nehmen sämtliche Komponenten Einfluss, denn der Zeitstempel errechnet sich aus der Zeitsynchronisation, aus der Position des Soundkarten-Puffers und aus dem Ergebnis der Signalanalyse. Das bedeutet, dass die erreichbare Genauigkeit höchstens so gut ist wie die schlechteste Komponente. Im weiteren Sinne ist ein berechneter Zeitstempel unbrauchbar, sobald nur eine einzige Komponente versagt.

Tatsächlich war es nicht trivial, jede Komponente in einem Multitasking-Betriebssystem wie Windows XP so zu trimmen, dass die Gesamtgenauigkeit von **ca. 0,5 ms** erreicht werden konnte. Verschiedene Techniken wurden in jedem Modul angewendet, die bestimmte Probleme lösten:

- Erst der Einsatz der linearen Regression in der **Zeitsynchronisierung** vermied durch hohe Pingzeiten verursachtes Nachschwingen. Die Präzision liegt bei **ca. 0,1 ms**, was durch Vorhersage der erwarteten Zeit bei der Synchronisation gemessen wurde.
- Das **Rauschen der Soundkartenposition** konnte durch den Einsatz des ASIO-Soundkartentreibers *ASIO4ALL* verringert werden, die Glättung wird somit besser. Mit genau definierten Einstellungen bewirkt er zudem eine Eingrenzung des systematischen Fehlers durch die Soundkartenlatenz auf 3 ms, welche manuell zu ermitteln ist. Dennoch ergaben Tests eine verbleibende Standardabweichung von ca. 0,15 ms, so dass der Soundkartendrift mit **ca. 0,5 ms** die maßgebliche Fehlerquelle bleibt.
- Die **Signalanalyse** ist bei Bestimmung der steigenden Flanke am Signalanfang bei harten Signalen bis auf wenige Samples genau, das entspricht **etwa 0,1 ms** bei 44.100 Hz Samplingrate. Gemessen wurde dieser Wert durch Plots der Signalaufzeichnung, ähnlich Abb. 5.3. Bei weichen, leisen Signalen ohne klare Flanke ist die Genauigkeit schlechter und liegt im Bereich von Millisekunden.

In einer Vielzahl von Messungen und Experimenten wurden diese Werte untersucht und verbessert. Die angefallenen Datenmengen sind groß und beinhalten eine Unzahl von Plots und Tabellen. Die interessantesten wurden bereits in den vorangegangenen Sektionen erläutert oder finden sich im Experimental-Kapitel (Kapitel 6). Deshalb wird an dieser Stelle nur noch das Experiment zur Messung der Schallgeschwindigkeit vorgestellt.

### 5.5.2 Exkurs: Messung der Schallgeschwindigkeit

Die Timestamping-Software lässt sich für das Rückwärts-Experiment verwenden: Anstatt aus einer gegebenen Schallgeschwindigkeit  $c$  und dem Laufzeitunterschied  $\Delta t$  den Distanzunterschied zu berechnen ( $\Delta d' = \Delta t \cdot c$ ), kann man auch die Distanz vorgeben und aus dem Laufzeitunterschied die Schallgeschwindigkeit berechnen ( $c' = \Delta d / \Delta t$ ).

Hierfür wurden dieselben zwei Rechner so aufgebaut, dass ihre Mikrofone einen Distanzunterschied von  $\Delta d$  zur Schallquelle hatten. Ein Schallsignal wurde abseits dieser beiden Rechner erzeugt, so dass die Schallquelle und die beiden Mikrofone eine Linie bildeten. Zunächst wurde das Timestamping auf beiden Rechnern gestartet und einen Moment abgewartet, damit sich die Rechner einpendeln konnten (Soundkarten-Drift und Zeitsynchronisierung). Dann wurden knapp 100 Schallsignale im Abstand von etwa einer Sekunde erzeugt. Anschließend wurde der mobile Rechner in einem Kontrollexperiment so platziert, dass eine Schallquelle gleichen Abstand zu beiden Mikrofonen hatte. Es wurden noch einmal 30 Schallsignale erzeugt.

Bei der Auswertung wurden zunächst alle Timestamps beider Messreihen eliminiert, die nicht zum Experiment oder zur Kontrolle gehörten, oder keinen gültigen Gegenpart auf dem anderen Rechner hatten. Diese waren aufgetreten, da der mobile Rechner bei laufender Messung herumgetragen wurde, und durch Umgebungsgeräusche. Nach Elimination der ungültigen Timestamps blieben 92 Experimentalwerte und 29 Kontrollwerte übrig.

Der Distanzunterschied im Experiment lag bei  $\Delta d = 11,4$  m mit 0,05 m Genauigkeit. Man kann leicht nachrechnen, dass eine Fehlmessung der Distanz um 0,1 m ( $\approx 1\%$ ) das Ergebnis um ca. 3 m/s verfälschen würde. Eine größere Distanz würde die Messgenauigkeit erhöhen.

Das Experiment ergab eine mittlere Zeitdifferenz von  $\Delta t = 31,62$  ms (0,60 ms), der Wert in Klammern bezeichnet die Standardabweichung. Bei der Kontrolle wurde eine Zeitdifferenz von  $\Delta t_k = -1,40$  ms (0,28 ms) ermittelt. Sie sollte eigentlich 0 ms sein, wegen der verschiedenen Hardware ist ein solcher Fehler aber nicht überraschend. Die geringe Standardabweichung der Kontrolle zeigt, dass der Fehler signifikant ist. Da die Messung zwischen Kontrolle und Experiment nicht unterbrochen wurde, hat er auch während des Experiments bestanden, man kann ihn also abziehen. Die korrigierte Zeitdifferenz ist 33,02 ms. Aus der Zeitdifferenz und dem Distanzunterschied ergibt sich

$$c' = \frac{\Delta d}{\Delta t - \Delta t_k} = 345,3 \text{ m/s} \quad (6,2 \text{ m/s}). \quad (5.20)$$

Das trifft relativ genau die reale Schallgeschwindigkeit von 346,3 m/s in Luft (bei 25 °C). Man sollte sich natürlich nicht von dem exakten Ergebnis blenden lassen, ohne die Standardabweichung zu berücksichtigen. Dennoch konnte gezeigt werden, dass die vom Timestamping ermittelten Werte plausibel und vergleichsweise genau sind.

## 5.6 Soundkartenlatenz

Eine Schwierigkeit die bereits im Vorfeld vermutet wurde, war die Latenz der Soundkarte. Dies ist die Zeit zwischen dem Eintreffen eines Schallimpulses am Mikrofon und dem Zeitpunkt, wann er in den Speicher geschrieben wird. Unterscheidet sie sich bei verschiedenen Rechnern, so führt das zu verschiedenen Zeitstempeln trotz identischem Signalzeitpunkt am Mikrofon.

Im Experiment mit dem Desktop und dem Laptop war zunächst bei identischem Abstand der Schallquelle zu zwei Mikrofonen nur eine geringe systematische Differenz von 0,5 ms zwischen beiden resultierenden Timestamps zu messen. Das Audiosignal wurde also bei beiden Rechnern zur selben Zeit verarbeitet. Deshalb wurde vermutet, dass beide Soundkarten entweder dieselbe Latenz besitzen, oder dass die oben genannten Verzögerungen nicht in dem Maße für den analogen Eingang gelten. Dies erschien plausibel, denn das analoge Signal wird über den Hardware-Analog-Digital-Wandler in ein

digitales Samplmuster verwandelt und anschließend in den Pufferspeicher geschrieben. Das alles geschieht auf Hardwareebene mit nur geringer Verzögerung.

Aus diesem Grunde wurde der Hardware-Latenz zunächst keine Bedeutung beigemessen, zumal der sonstige statistische Fehler noch bei über 10 ms lag. In späteren Experimenten kamen neue Rechner hinzu und zeigten in Feldversuchen mit bis zu 6 Rechnern systematische Differenzen von über 50 ms, einer Ewigkeit im Vergleich zu den eigenen Messungen. Der Feldversuch fand in einem Hörsaal der Albert-Ludwigs-Universität statt, die Rechner waren über LAN miteinander verbunden. Die Gründe für den großen Fehler sind nicht bekannt. Spätere eigene Versuche konnten derartige Fehler nicht reproduzieren.

Als der statistische Fehler durch Verbesserung der Implementation mit der Zeit auf unter 1 ms sank wurde deutlich, dass der systematische Fehler sehr wohl bestand, allerdings nicht in dem Maße, wie er im Feldversuch aufgetreten war. Nachdem der ASIO-Treiber zum Einsatz kam, wurde erneut gemessen. Auf insgesamt sieben Geräten lag der Latenzunterschied im Bereich von  $\pm 4$  ms zum Referenzrechner, dem Desktop.

In einem Experiment wurde versucht die Latenz einer Soundkarte zu messen. Ein kurzes, hartes Signal wurde abgespielt und der Zeitpunkt des Kommandos dafür wurde festgehalten. Dieses Signal wurde von der gleichzeitig laufenden Timestamping-Engine aufgezeichnet und der Zeitpunkt bestimmt. Es ergaben sich Latenzen zwischen 50 und 150 ms. Problematisch an der Methode ist jedoch, dass nicht die Aufnahme-Latenz selbst gemessen wird, sondern die Dauer für Abspielen und Aufnehmen zusammen. Beide sind vermutlich nicht identisch, so dass man nicht einfach durch zwei teilen kann. Desweiteren erscheinen die erzielten Werte viel zu groß. Das Experiment war vermutlich ungeeignet die Latenz zu messen. Offenbar kann die Latenz der Soundkarte nicht ohne komplizierte Zusatzhardware gemessen werden. Man würde einen Geräuschgenerator benötigen, der ein Schallsignal unmittelbar neben dem PC-Mikrofon zu einem definierten Zeitpunkt erzeugt. Das heißt, dass die Latenz des Zusatzgerätes bekannt sein müsste. Dann könnte man den Empfangszeitpunkt an der Soundkarte durch Anzeige des Zeitstempels messen. Auch der zeitliche Offset zwischen dem Geräuschgenerator und dem PC müsste bekannt sein.

Ohne diese Zusatzhardware kann nur eine relative Latenz im Vergleich zu einem anderen PC bestimmt werden, was für unsere Belange völlig ausreicht. Das bedeutet, dass jeder neu verwendete Rechner einmalig kalibriert werden muss. Referenz ist hierfür der Desktoprechner mit  $T_{\text{latency}} = 0,0$  ms, alle anderen Rechner sind relativ dazu. Wie man leicht einsehen kann, muss als Vergleichsrechner nicht der Desktoprechner verwendet werden, jeder bereits kalibrierte Rechner ist hierfür geeignet.

Problematisch ist, dass der Offset bei manchen Rechnern nicht stabil ist, sondern zwischen verschiedenen Experimenten im Bereich von 1 ms schwankt, etwa beim Laptop *Amilo Pro*. Der Rechner muss jedesmal erneut kalibriert werden, und selbst dann ist

nicht sicher, dass der Wert im Laufe der Messung stabil bleibt. Bei den meisten Rechnern ist die ermittelte Latenz in jedem Versuch gleich. Man kann eine Unterscheidung in geeignete und weniger geeignete Rechner treffen.

## 5.7 Weiterführende Signalverarbeitung

Die momentane Methode den Zeitpunkt der Schwellwertüberschreitung zu verwenden, ist bei lauten Signalen mit klar definierter Flanke, etwa dem Schlagen eines Löffels gegen eine Glasflasche, ziemlich genau, jedoch noch nicht optimal. Ein alternativer Ansatz versucht die Position der Flanke direkt zu ermitteln. Hierfür wird die rechteckige Fläche vom Anfang des detektierten Signals  $T_0$  bis zum RMS-Maximum der Amplitude  $T_{\max}$  bestimmt und ein Wert  $k$  so optimiert, dass gilt:

$$k_{\max} \leftarrow \arg \max_k \int_{T_0}^{T_k} y(T_{\max}) - y(T_i) \, di + \int_{T_k}^{T_{\max}} y(T_i) \, di \quad (5.21)$$

Anschaulich gesprochen wird die „Signalfläche“ vor der Flanke minimiert und nach der Flanke maximiert. Durch Speicherung von Zwischenergebnissen der diskreten Integration kann die Optimierung in linearer Zeit erfolgen. Das funktioniert bei eindeutigen Flanken sehr gut und führt zu genaueren Zeitstempeln als die Verwendung des Zeitpunktes der Schwellwertüberschreitung, welcher vor der Flanke liegen kann. Bei weichen Signalen, die den Schwellwert nur mühsam überschreiten, lässt sich jedoch keine Flanke bestimmen. Der Optimierungsalgorithmus macht hier sogar größere Fehler als die Schwellwertüberschreitung.

Denkbar ist eine Verbesserung durch direkten Vergleich zweier Signale untereinander. Sie werden gegeneinander verschoben, bis eine maximale Ähnlichkeit erreicht ist. Konkret geschieht das durch Kreuzkorrelation beider Signale mittels inverser Fouriertransformation über alle Verschiebungen, wodurch eine optimale Verschiebung  $k$  mit maximaler Korrelation bestimmt wird. Eine Variante dieses Ansatzes wurde von Valin et al. verwendet [7]. Der Nachteil dieser Methode ist, dass die Signale zwischen den Rechnern ausgetauscht werden müssen, was das Netzwerk belastet. Zudem müssen genauso viele Signale miteinander verglichen werden, wie Rechner an der Ortung teilnehmen. Bei der autarken Bestimmung der Zeitstempel ist das nicht der Fall.

Bei Verwendung einer Differenzialmethode wurden ähnliche Mehrdeutigkeiten festgestellt wie bei der RMS-Maximumsuche. Der Zeitpunkt des maximalen Signalanstiegs kann bei einem Signal auf zwei verschiedenen Rechnern an zwei völlig verschiedenen Stellen liegen. Aus diesem Grunde wurde bislang der robuste Schwellanstiegszeitpunkt beibehalten.

## 5.8 Zusammenfassung

Je genauer das Timestamping den Zeitpunkt von auftretenden Schallsignalen bestimmen kann, desto weniger Anforderungen werden an Korrekturalgorithmen der Lokalisierung gestellt. Deswegen ist ein wichtiges Ziel, die Genauigkeit des Timestamping so weit wie möglich zu steigern.

Die Verwendung hochwertiger Soundkarten könnte eine Verbesserung bringen. Diese besitzen genaue, eigene Uhren und hätten so eine bessere Laufruhe, als die billigen Chips der verwendeten Onboard-Soundkarten. Ihr Treiber könnte eine exakte Position des Hardware-Puffers anzeigen, anstatt in Blöcken von 512 Samples zu springen – und das in unvorhersagbaren Sprüngen von 1 oder 2 Blöcken, was fehleranfällig geglättet werden muss. Da gute Soundkarten eigene Prozessoren besitzen, würde die Aufzeichnung auch nicht, wie im Experiment beobachtet, zu driften beginnen, wenn die CPU ausgelastet ist, sondern wäre unabhängig von der Systemlast.

Zukünftig soll eine Verwendung der Schallortung auf spezialisierten Systemen geprüft werden. Diese könnten geeignetere Hardware und Software zur Verfügung stellen, als es ein handelsüblicher PC oder Laptop kann. So existieren echtzeitfähige Linux-Kernel, z.B. RTAI [20] oder RTLinux, die in Verbindung mit Prozessoren zur Signalverarbeitung (DSP) verwendet werden könnten.

Es ist bemerkenswert, dass bei der enormen „Hardware-Diversifikation“ überhaupt eine solche Genauigkeit erreichbar ist, wie sie im Experiment gezeigt wurde. Vermutet wurde bei Projektbeginn eine Präzision im Bereich von Hundertstelsekunden. Das lag schon allein daran, dass die standardmäßige Uhr, die von der C++-STL zur Verfügung gestellt wird, eine Schrittweite von 16 ms hat, einer Ewigkeit im Vergleich zu den Werten, die für die Raumortung notwendig sind und die hier bereitgestellt werden. Kann jetzt noch die Robustheit des Timestamping verbessert werden, so steht einem Feldeinsatz der Software nichts mehr im Wege.

## 6 Komponentenintegration

Mit dem *Lampshade*-Algorithmus wurde eine Möglichkeit geschaffen, aus gegebenen Timestamps die Position von Signalquellen zu bestimmen, selbst wenn die Mikrofonpositionen nicht bekannt sind. Die Software *BasicTimestamping* erzeugt solche Zeitstempel und speichert sie in einer Datei ab, oder sendet sie an einen Master-Host. Es ist naheliegend, die beiden Module in einer Software zu kombinieren. Dieser Vorgang wird zweistufig ablaufen.

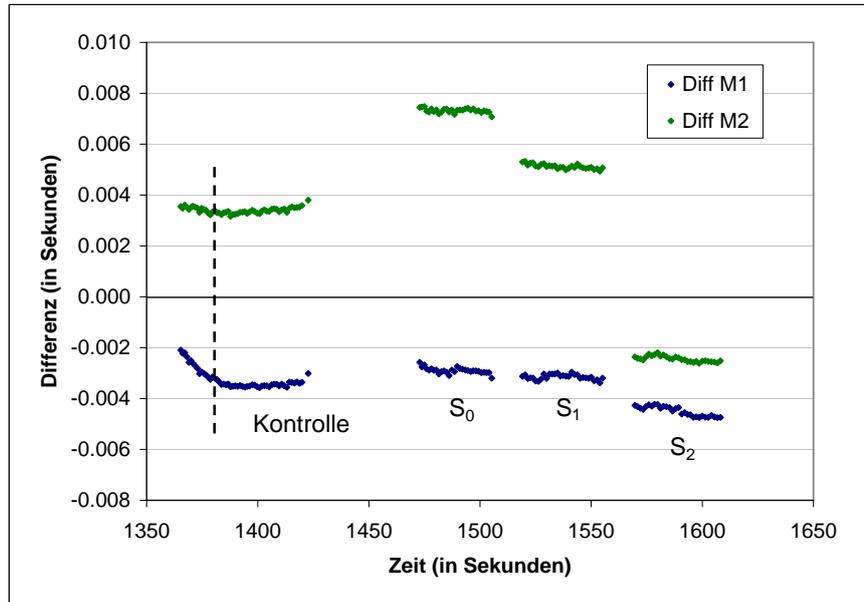
In der ersten Stufe wird eine *Offline*-Ortung realisiert: Mit *BasicTimestamping* werden Daten gesammelt und nach der Aufzeichnung in Textdateien gespeichert. Anschließend werden die Dateien von den verschiedenen Rechnern eingesammelt, miteinander assoziiert, von Hand gefiltert und als Eingabewerte für den *Lampshade*-Algorithmus verwendet. Dieser versucht den Rückschluss auf die Position der Schallquellen- und Mikrofonpartikel.

In der zweiten Stufe soll die Assoziation der Zeitstempel zu logischen Schallquellen automatisch geschehen. Nachdem ein Timestamp aufgezeichnet wurde, wird er an den Master-Host gesendet und gefiltert. Unmittelbar danach wird die Lokalisierung ausgeführt und auf dem Bildschirm angezeigt – eine „echte“ *Online*-Lokalisierung. Sie ist weitaus komplizierter und setzt das perfekte Zusammenspiel der beiden Komponenten voraus, da die manuelle Filterung der Eingabedaten wegfällt und keine Mittelung mehrerer Zeitstempel möglich ist.

### 6.1 Offline-Experiment

Zwar hatte eine Simulation mit Fehlermodell schon Hinweise darauf gegeben, dass sich reale Zeitstempel als Eingabe für die Lokalisierung eigneten, dies war jedoch experimentell zu belegen. Ein einfaches 2D-Szenario wurde aufgebaut: Die Positionen von drei Mikrofonen  $M_0$  bis  $M_2$  waren gegeben und es sollten die Positionen von drei Schallquellen  $S_0$  bis  $S_2$  bestimmt werden.

In diesem frühen Experiment war die Kompensation der Hardwarelatenz  $T_{\text{latency}}$  noch nicht in *BasicTimestamping* implementiert. Deshalb wurde zunächst ein Kontrollexperiment durchgeführt, wobei alle Mikrofone von einer Schallquelle gleich weit entfernt waren. Zum Einsatz kamen ein Desktop und zwei Laptops der 2-GHz-Klasse. Der Desktop war per LAN-Kabel an das lokale Netz angeschlossen und verfügte über ein



**Abbildung 6.1:** 2D-Experiment mit 3 Mikrofonen und 3 Schallquellen: Relativer Zeitunterschied der Signale an den Mikrofonen  $M_1$  (blau) und  $M_2$  (grün) im Vergleich zu Mikrofon  $M_0$ . Von der Kontrollreihe wurde nur der nicht driftende Bereich verwendet.

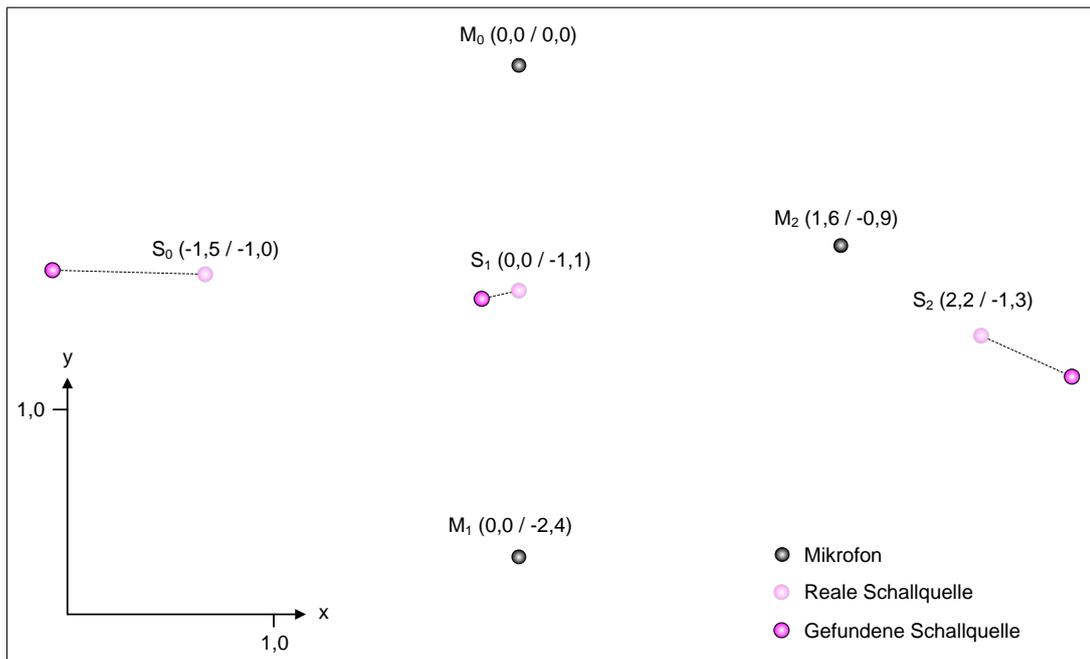
angeschlossenes Ansteck-Mikrofon, die Laptops wurden per WLAN eingebunden und besaßen ein eingebautes Mikrofon. Die Signale wurden mit einem Löffel erzeugt, indem im Sekundenabstand 30 mal gegen eine Glasflasche geschlagen wurde. Die charakteristische Form dieser Signale ist geeignet, da sie eine sehr steile Anfangsflanke besitzen. Mit dieser Flanke kann der Zeitpunkt des Signals besonders genau bestimmt werden. Die Kontrolle stellte sicher, dass ein konstanter Offset zwischen den Rechnern ermittelt und später herausgerechnet werden kann.

Danach wurden die Mikrofone auf einer Ebene im Raum an vorgegebenen Positionen platziert. An drei verschiedenen Stellen wurden Reihen von Schallsignalen erzeugt und die Positionen markiert. Insgesamt ergaben sich eine Kontrollreihe und drei Reihen von Messdaten. Jeder Messblock wurde gemittelt und anschließend der Mittelwert der Kontrolle abgezogen.

Nachdem die Signale aufgenommen waren, wurden die Distanzen  $d$  zwischen den Signalpositionen und Mikrofonen mit einem Meterstab auf einen Dezimeter genau gemessen. Das Mikrofon  $M_0$  wurde als Ursprung eines Koordinatensystems gewählt, die Positionen der anderen Objekte wurden mit Zirkel und Geodreieck geometrisch bestimmt. Dies kann man auch numerisch lösen, etwa mit einem kurzen Maple-Skript [21]. Nun wurden die Zeitstempel und die realen Mikrofonpositionen in die Lokali-

Objekt	$M_1$	$M_2$
Kontrolle	-3.456 (0.099)	3.380 (0.122)
$S_0$	-2.900 (0.120)	7.320 (0.088)
$S_1$	-3.154 (0.104)	5.124 (0.093)
$S_2$	-4.499 (0.194)	-2.444 (0.114)
$S_0$ (korr.)	0.557	3.940
$S_1$ (korr.)	0.303	1.744
$S_2$ (korr.)	-1.043	-5.825

**Tabelle 6.1:** 2D-Experiment mit 3 Mikrofonen und 3 Schallquellen: Durchschnittswerte der relativen Zeitstempel bezüglich  $M_0$  in ms vor und nach Abzug der Kontrolle. Der Wert in Klammern bezeichnet die Standardabweichung.



**Abbildung 6.2:** Maßstabsgetreue Positionen der tatsächlichen (blassviolett) und der gefundenen Schallquellen (violett). Die Mikrofone sind grau eingezeichnet. Alle Einheiten in Metern.

sierung eingetragen. Wie in Abschnitt 4.4 erläutert, sind der Physiksimulation diese Positionen zu keinem Zeitpunkt bekannt. Die Simulation wurde wie in den Versuchen in Kapitel 3 ausgeführt, bis sie eine Abbruchbedingung erreicht hatte oder wegen Erreichen der maximalen Zyklenzahl terminierte.

In Abbildung 6.2 sieht man, dass die gefundenen Positionen von den tatsächlichen Schallquellen-Positionen zwar abweichen, aber tendenziell an der richtigen Stelle ermittelt wurden. Die zentrale Schallquelle  $S_1$  scheint sich dabei besser lokalisieren zu lassen, als die außerhalb des Perimeters der Mikrofone liegenden Schallquellen  $S_0$  und  $S_2$ . Der Grund für die horizontale Abweichung – weg von  $M_1$  – ist vermutlich eine schlechte Kalibrierung des Mikrofons  $M_1$ , es zeigt einen zu späten Zeitstempel an. Die geringe Standardabweichung der Messreihen lässt darauf hoffen, dass sich die Ortung auch im Innenraum mit begrenzten Dimensionen und damit hoher, benötigter Genauigkeit durchführen lässt, falls man den konstanten Offset von mehreren Millisekunden zu beherrschen lernt. Die erreichten 0,1 ms suggerieren, dass 95 % aller Positionen einer Messreihe normalverteilt in einem Kreis mit 7 cm Radius liegen, was für eine Innenraum-Ortung ausreichen sollte. Zuletzt sollte auch die ungenaue Ermittlung der Schallquellen-Positionen mit einem Meterstab angemerkt werden. Die genaue Bestimmung des Schallursprungs von genauer als einem halben Dezimeter ist nicht einfach und begrenzt die erzielbare Ortungsgenauigkeit.

Dieses Experiment ist nur begrenzt geeignet, quantitative Aussagen über die Lokalisierbarkeit der Schallquellen zu machen. Eine Schallquelle wurde genauer geortet, die anderen weniger genau. Der Grund dafür lässt sich nur noch erraten, nicht numerisch ausdrücken. Zu dem Zeitpunkt, als der Versuch erdacht wurde, war vielmehr die grundsätzliche Frage zu klären, ob die erzeugten Zeitstempel als Eingabe für die Lokalisierung geeignet sind und ob die Lokalisierung in so kleinem Maßstab in einem Zimmer mit 14 qm durchführbar ist. Beide Fragen konnten mit „ja“ beantwortet werden. Der nächste Schritt ist die „Live“-Ortung empfangener Zeitstempel.

## 6.2 Live-Ortung mit bekannten Mikrofonpositionen

Das vorangehende Experiment ließ erwarten, dass Zeitstempel auch einzeln, ohne Mittelung, zur Lokalisierung herangezogen werden können. Würden die Zeitstempel nicht nur in einer Datei lokal gespeichert, sondern sofort an einen *Master*-Host übermittelt, so könnte unmittelbar der Algorithmus zur Lokalisierung ausgeführt werden.

Hierzu wurde eine Integration der beiden Softwaremodule in ein Programm *CompleteLocalization* vorgenommen. Wegen des modularen Aufbaus der Klassen, musste nur die Anwendungsklasse `Application` neu geschrieben werden, die inhaltlichen Komponenten `Simulation`, `TimestampRecorder` und ihre Unterklassen konnten unverändert

bleiben. Zur Steuerung der Unterklassen wurden jedoch einige Schnittstellen hinzugefügt. Technisch bindet *CompleteLocalization* den Code der Basisprojekte *BasicLocalization* und *BasicTimestamping* ein, so dass Weiterentwicklungen der kombinierten Software auch den Basisprojekten zugute kommt.

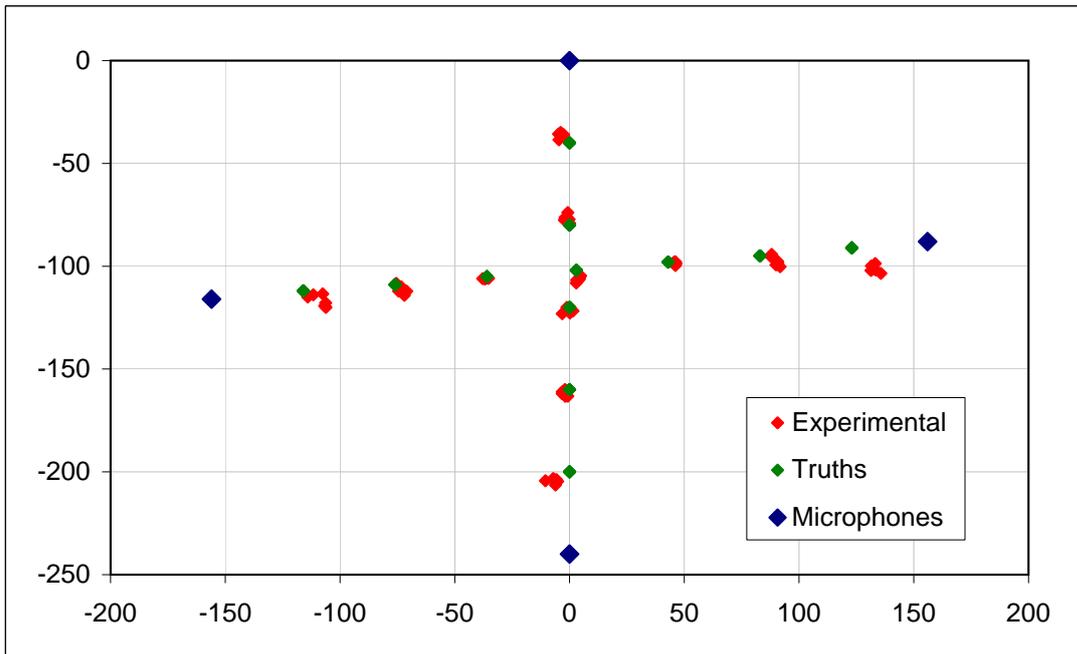
### 6.2.1 Versuchsaufbau

Die in die Lokalisierung involvierten Rechner werden über Netzwerk miteinander verbunden, wobei auch ein stabiles WLAN verwendet werden kann. Einer wird als Master-Rechner bestimmt, auf ihm wird *CompleteLocalization* ausgeführt, auf allen anderen Rechnern läuft *BasicTimestamping*. Empfängt ein Rechner ein Schallereignis, so wird der ermittelte Timestamp per Netzwerk sofort an den Master-Rechner weitergeleitet. Nachdem der Master-Rechner einen neuen eigenen oder fremden Timestamp erhalten hat, beginnt er eine Wartephase von 200 ms. In dieser Zeitspanne eintreffende Timestamps gelten als zusammengehörig. Wurden von allen Rechnern im Netzwerk Timestamps empfangen, so wird in der Simulation eine logische Schallquelle erzeugt, und mit Constraints versehen. Die Zuordnung ist eindeutig, da die Namen der Rechner im Netzwerk bekannt sind.

Zunächst wurde ein 2D-Experiment mit vier bekannten Mikrofonpositionen, also vier Rechnern durchgeführt. Sie wurden, wie in Abschnitt 6.1, an bestimmten Raumpositionen aufgestellt und die Position in die Software des Master-Rechners eingetragen. Es wurden dieselben Positionen wie im letzten Experiment gewählt, hinzu kam jedoch der vierte Rechner. Die größere Anzahl an Mikrofonen stabilisiert die Lokalisierung.

Nun wurde ein Schallquellen-Setup mit bestimmbareren Schallereignis-Positionen entworfen. Hierfür wurden zwischen je zwei gegenüberliegenden Mikrofonen Bindfäden mit Markierungen im Abstand von 40 cm gespannt, so dass sich beide Fäden etwa in der Mitte kreuzten, siehe Abbildung 6.3. Die 2D-Positionen der Markierungen wurden berechnet und in die Auswertung eingetragen. Anschließend wurde die Aufnahme gestartet und an jeder Markierung wurden 6 Schallereignisse erzeugt.

Die Rechner waren zuvor sorgfältig kalibriert worden, um verbleibende systematische Fehler auszuschließen. Ursächlich für diese Fehler ist die unterschiedliche Hardware eines jeden Gerätes, deren Latenz nicht bekannt ist und daher nur experimentell kompensiert werden kann. Wie in Abschnitt 6.1 erfolgt die Kalibrierung durch Aufstellung der Rechner in einer Reihe, so dass alle Mikrofone von einer Schallquelle gleich weit entfernt sind. Die Timestamps sollten in dem Fall identisch sein, der dennoch zu beobachtende Unterschied ergibt den Offset des systematischen Fehlers. Er liegt in der Größenordnung von  $\pm 3$  ms zur Referenz.



**Abbildung 6.3:** 2D-Draufsicht der Positionen von Ground-Truth-Daten (grün), Experimentaldaten (rot) und Rechnern (blau). Die Skaleneinheit ist *cm*. Bemerkenswert ist der zunehmende systematische Fehler am Rande des Perimeters.

### 6.2.2 Ergebnis

Die berechneten Positionen der Schallsignale und die mit den Bindfäden ermittelten Positionen (*Ground-Truth*-Daten) wurden in einem 2D-Plot aufgetragen, vgl. Abb. 6.3. Zwischen dem Mittelwert der Experimentaldaten und der *Ground-Truth*-Position wurde die euklidische Distanz berechnet. Die Distanz und die Standardabweichung der Messung sind in Tabelle 6.2 zu sehen.

Die Ergebnisse zeigen einen geringen Fehler und eine genaue Lokalisierung innerhalb des Perimeters, der durch die vier Mikrofone umrandet wird. Nach vorangegangener Kalibrierung liegt die Distanz der Messdaten von den *Ground-Truth*-Daten meistens im Bereich von unter 10 cm. Die Standardabweichung der Messungen liegt im Bereich von  $\sigma = 2$  cm, wobei die geringe Zahl von 6 Schallsignalen je Position anzumerken ist.

Das Resultat dieses Versuchs ist vielversprechend. Die Schallortung ist bei vorgegebenen Mikrofonpositionen im kleinen Maßstab eines Innenraumes durchführbar – zumindest wenn das Szenario günstig gewählt ist. Die Schallquellen sollten im Perimeter der Mikrofone liegen, aber einen gewissen Mindestabstand zum nächsten Mikrofon nicht

Index	Distanz	$\sigma_x$	$\sigma_y$
1	8.6	3.3	2.8
2	3.4	1.7	2.0
3	1.1	1.1	0.2
4	4.5	0.7	1.1
5	3.1	0.3	0.7
6	7.2	1.6	2.3
7	14.1	1.6	1.7
8	8.2	1.9	1.1
9	2.9	0.9	1.3
10	1.6	1.6	1.2
11	3.4	0.9	1.7
12	5.3	0.8	1.3

**Tabelle 6.2:** Distanz und Standardabweichung des 2D-Online-Experiments mit 4 Rechnern in *cm*. Die Indizes entsprechen den Datenpunkten aus Abb. 6.3 von *links* nach *rechts* und anschließend von *unten* nach *oben*.

unterschreiten. Frühere Versuche zeigten große systematische Schwierigkeiten des Systems, wenn Schallereignisse unmittelbar neben Mikrofonen auftraten. Sie sind in Abb. 6.3 ansatzweise zu erkennen, etwa die erhöhte Distanz der Rand-Indizes 1, 7, 8 und 12. Im günstigen Fall führt die geringe Standardabweichung der Kontrollmessungen mit Streuungen von ca. 0,1 ms auf eine geringe Streuung der Lokalisierung von 3 cm, was dem Raumäquivalent der zeitlichen Streuung entspricht.

### 6.2.3 Diskussion

Während der statistische Fehler bei stabilem Netzwerk (Zeitsynchronisierung) und harten Signalfanken gering ist, muss jedoch einiges an Aufwand betrieben werden, um den systematischen Fehler in Grenzen zu halten. Der konstante Offset der Soundkarte kann zwar bestimmt werden, es scheint aber noch eine zweite Fehlerquelle zu geben, die eine Schwankung im Bereich von 1 ms verursacht. Vermutlich kommen weniger geeignete Computer mit schlechten Soundchips hierfür in Betracht. Deshalb muss am Versuchsanfang erneut kalibriert werden, andernfalls ergab sich eine systematische Verschiebung der Positionen um etwa 20 cm.

Desweiteren können sich Fehler durch ungenaue Vermessung des Experimental-Aufbaus ergeben. Durch Messen der Distanzen mit einem Meterstab und mit dem markierten Bindfaden, konnte die Genauigkeit des Experimental-Setups mit ca. 3 cm angegeben werden. Durch noch sorgfältigeres Arbeiten könnte man noch ein wenig genauer werden, aber mit den verwendeten Mitteln Meterstab und Glasflasche als Signalerzeuger

ist der Präzision der Ground-Truth-Daten bei 2 cm eine Grenze gesetzt. Bei der Versuchsdurchführung ist darauf zu achten, dass die Mikrofone möglichst nicht durch den eigenen Körper verdeckt werden. Der Umweg, den der Schall nehmen muss, und die Abschwächung des Signals durch den Körper, führen zu einer Fehlmessung am verdeckten Mikrofon.

Ohne Vorverarbeitung der eingehenden Zeitstempel konnte dieses Experiment erfolgreich durchgeführt werden. Beim Durchsehen der vielen möglichen Fehlerquellen, von denen jede einzelne zu einem Totalausfall der Ortung führt, wird jedoch klar, dass die Robustheit des Systems noch einiger Verbesserungen bedarf. Erst dann lässt es sich auch unter Feld- bzw. Outdoorbedingungen verwenden.

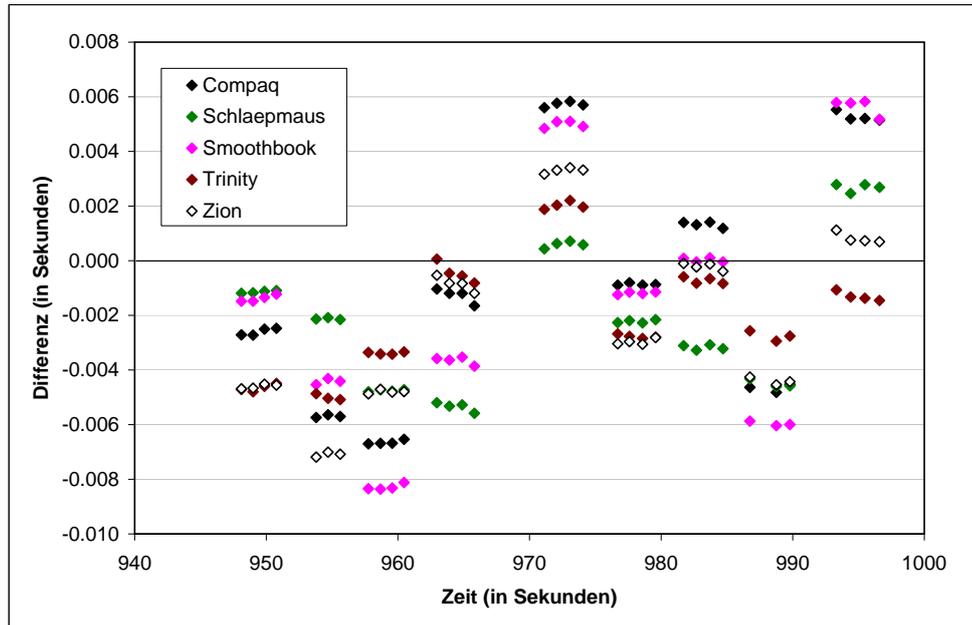
## 6.3 Ortung mit unbekanntem Mikrofonpositionen

Thematisch zur Offline-Lokalisierung passend, aber chronologisch nach den anderen Experimenten liegend, wird das Experiment zur Ortung unbekannter Mikrofone als letztes vorgestellt. Vom Effekt her ist es weniger imposant als die vorangegangene Live-Ortung, doch inhaltlich stellt es einen Höhepunkt dieser Experimentalreihe dar.

Die Ortung von Schallquellen mit Hilfe *bekannter* Mikrofone stellt für den entwickelten Lampshade-Algorithmus einen einfachen Fall dar. Die Iteration führt bei plausiblen Inputdaten stets zu sinnvollen Ergebnissen, soweit die experimentelle Beobachtung. Sind die Timestamps eines Schallereignisses fehlerhaft, so schlägt diese eine Ortung fehl, andere Ortungen bleiben davon unberührt. Die Positionen der Schallquellen sind nur von den festen, korrekt vorgegebenen Mikrofonpositionen abhängig.

Bei der wechselseitigen Lokalisierung mit *unbekannten* Mikrofonen hingegen entsteht ein kompliziertes Netzwerk wechselseitiger Abhängigkeiten. Schallquellen-Positionen sind zwar nicht direkt miteinander verbunden, jedoch indirekt über die Mikrofonpositionen. Das bedeutet, sollte es gelungen sein, ein intaktes Netzwerk von Schallquellen und Mikrofonen zu berechnen, und eine weitere Schallquelle mit fehlerhaften Messdaten wird hinzugefügt, so beeinträchtigt sie die Positionen der Mikrofone. Diese wiederum verderben die Positionen der anderen Schallquellen.

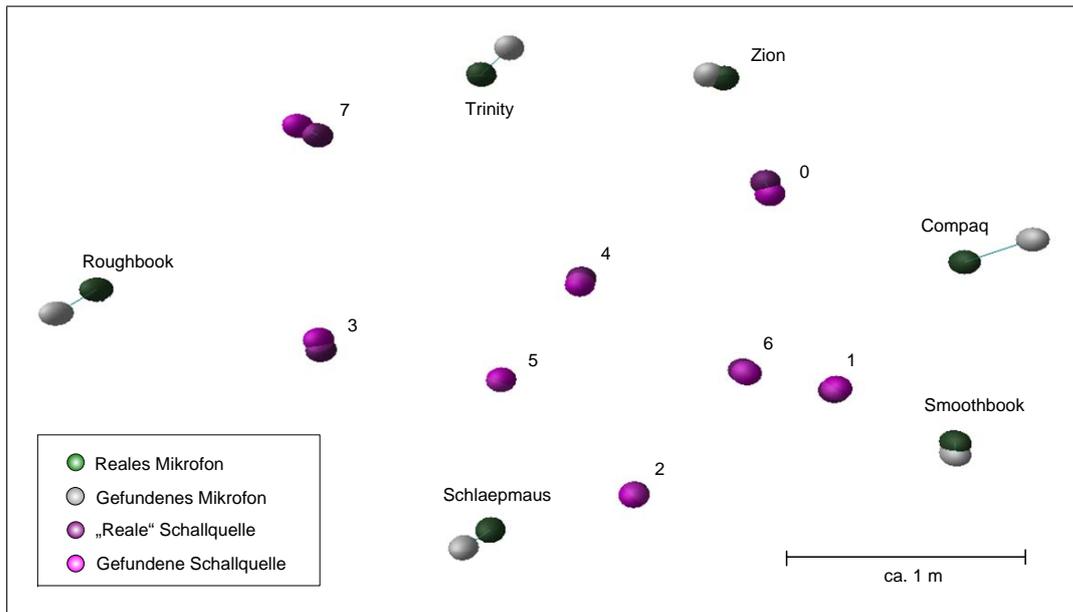
Um diese Situation zu verhindern, werden umfangreiche algorithmische Gegenmaßnahmen notwendig sein, die eingehende Timestamps vorfiltern und die Qualität des Netzes vor und nach dem Hinzufügen vergleichen. Die Entwicklung dieser Algorithmen an sich stellt keine besondere Hürde dar, jedoch erfordert sie genaue Beobachtung der Szenarien und Erfahrung, wie Ausreißer sinnvoll bekämpft werden können. Aus diesem Grunde wird derzeit darauf verzichtet, eine Live-Ortung unbekannter Mikrofone zu versuchen, die momentan in Einzelfällen funktionieren kann, jedoch häufig so instabil sein wird, dass die Resultate unvorhersehbar sind. Dies sei eine Aufgabe für die Zukunft.



**Abbildung 6.4:** Zeitliche Differenz der Timestamps von 6 Rechnern auf einer Ebene. Aufgezeichnet wurde eine Serie von  $9 \times 4$  Schallsignalen. Die Zeitstempel sind relativ zum Rechner *Roughbook*.

Vielmehr soll gezeigt werden, dass eine Lokalisierung unbekannter Mikrofone mit echten Eingangsdaten grundsätzlich möglich ist, unter der Voraussetzung dass diese Daten manuell gefiltert wurden, wie es auch ein Algorithmus könnte. Das Experiment hierzu wurde auf ähnliche Weise durchgeführt wie die vorangehenden. Fünf Laptops und ein stationärer Desktop-PC wurden über WLAN bzw. LAN zu einem lokalen Netzwerk verbunden. In einem Raum der Größe von ca.  $4 \text{ m} \times 3 \text{ m}$  wurden sie ungefähr elliptisch auf einer Ebene angeordnet und ihre 2D-Position mit einem Meterstab auf 2 cm genau gemessen.

Auf allen Rechnern wurde die Software *BasicTimestamping* gestartet, auf dem Desktop-Rechner hingegen *CompleteLocalization*. Bei diesem waren die Mikrofonpositionen in die Software eingetragen, so dass Schallsignale visuell auf dem Bildschirm sichtbar wurden. Nach kurzen Schwierigkeiten mit der Stabilität des WLAN, konnte eine Liveortung der Schallquellen, wie im vorangehenden Abschnitt, durchgeführt werden. Neben einer Menge von Ausschussdaten, die durch geräuschvolles Werken an den bereits aufzeichnenden Rechnern anfielen, wurde eine Serie von  $9 \times 4$  Signalen aufgezeichnet, vier jeweils am selben Ort. Die hohe Zahl an Rechnern und Schallquellen war bewußt gewählt, um Reserven zu haben, falls ein Rechner systematisch falsch misst, oder eine Schallquelle ungenaue Zeitstempel produziert und um die Wahrscheinlichkeit für einen Fehlschlag durch lokale Minima zu senken.



**Abbildung 6.5:** Wechselseitige 2D-Lokalisierung mit 6 Mikrofonen und 8 Schallquellen, mittels SVD transformiert. Die Schallquellen decken sich mit der Ground Truth besonders gut, was auch ein Artefakt der automatischen Positionsbestimmung sein kann. *Abbildung entstammt einem Screenshot von BasicLocalization, farblich nachbearbeitet und beschriftet.*

Von allen teilnehmenden Computern wurden die Textdateien mit den Timestamps eingesammelt und die jeweils zusammengehörigen mit einem Python-Skript [22] einander zugeordnet. Anschließend wurden sie manuell gefiltert, und auffällige Ausreißer, etwa solche mit über 20 ms Abweichung zueinander, wurden gelöscht. Dies betraf alle Timestamps vor und nach der Serie, sowie vier Zeitstempel der Serie. Die übrigen, positionsgleichen Timestamps wurden arithmetisch gemittelt, die resultierenden 36 Timestamps wurden in eine Inputdatei von *BasicLocalization* eingetragen. Die relativen Zeitstempel sind in Abbildung 6.4 zu sehen. Die geringe statistische Abweichung der Zeitstempel einer Position lässt wie im ersten Offline-Experiment hoffen, dass eine Mittelung mehrerer Schallsignale in Zukunft nicht notwendig sein wird.

Nun wurde unter Verwendung dieser Timestamps und den sechs Mikrofonpositionen eine Lokalisierung von neun variablen Schallquellen mit *bekanntem* Mikrofonpositionen durchgeführt, um Ground-Truth-Daten der Schallquellen zu erhalten. Auf diese Weise konnte eine umständliche manuelle Vermessung der Schallquellen-Positionen wie im vorangegangenen Experiment eingespart werden. Jenes hatte gezeigt, dass die absolute Positionsbestimmung der Schallquellen vergleichsweise gut funktioniert. Ground-Truth-Daten der Mikrofone und der Schallquellen waren nun verfügbar, auch sie wurden in das Inputfile eingetragen.

Schließlich wurde die Ortung erneut durchgeführt, diesmal mit der Simulation *unbekannten* Mikrofonpositionen. Das Ergebnis waren korrekt ermittelte Schallquellenpositionen, jedoch systematische, radiale Abweichung der Mikrofonpositionen um ca. 0,7 m nach außen weg. In der Textausgabe des Programms wurde ein erhöhter, verbleibender RMS-Fehler der ersten Schallquelle bezüglich der Mikrofone angezeigt – testweise wurde sie gelöscht. Das verbesserte Ergebnis der anschließend durchgeführten Simulation mit acht Schallquellen ist in Abb. 6.5 zu sehen. Bevor die Software das Netz grafisch darstellte, wurde, wie in Abschnitt 4.5 beschrieben, jeweils eine kongruente Transformation mittels SVD durchgeführt, um das gefundene Netz am Ground-Truth-Netz auszurichten.

Im Experiment wurde die Erfahrung aus dem vorangegangenen Abschnitt angewendet, Schallsignale nur innerhalb des Perimeters der Mikrofone zu generieren. Dadurch lassen sich bessere Ergebnisse erzielen, als wenn Schallquellen abseits der Mikrofone liegen. In simulierten Tests mit dieser Einschränkung wurde sichergestellt, dass dies für die wechselseitige Ortung keine Verschlechterung bedeutet, weder in der Genauigkeit, noch in der Erfolgsrate.

Im Schaubild (Abbildung 6.5) ist eine besonders genaue Übereinstimmung der experimentellen Schallquellen mit den Ground-Truth-Schallquellen zu sehen. Dies ist möglicherweise ein Nebeneffekt der automatischen Bestimmung: Ein Fehler bei der Ortung der Schallquellen könnte in beiden Versuchen gleichermaßen aufgetreten sein. Es ist jedoch keine systematische Verschiebung der Mikrofonpositionen zu erkennen, deswegen ist davon auszugehen, dass die Schallquellenpositionen plausibel sind.

Mit diesem Experiment konnte gezeigt werden, dass die wechselseitige Lokalisierung nicht nur mit perfekten, simulierten Daten wie in Kapitel 4 funktioniert, sondern auch mit realen, fehlerbehafteten Zeitstempeln. Die Vorverarbeitung der Zeitstempel geschah derart, wie sie auch ein Filteralgorithmus durchführen könnte. Es wurden lediglich auffällige Zeitstempel gelöscht. Auch das Entfernen von Schallquellen ist legitim, wenn man davon ausgeht, dass auch ein algorithmischer Filter Schallquellen entfernen kann, die den Gesamtfehler des Netzwerks erhöhen.

## 7 Ausblick

Mit dem Lampshade-Algorithmus konnte ein verhältnismäßig schneller und universaler Algorithmus zur wechselseitigen Positionsbestimmung unbekannter Schallquellen- und Mikrofonpositionen entwickelt werden. Es konnte gezeigt werden, dass das Problem der wechselseitigen Lokalisierung auf die vorgestellte Art und Weise lösbar ist. Mit handelsüblicher PC-Hardware, dem Lampshade-Algorithmus und der Timestamping-Software konnte innerhalb des Perimeters mehrerer Mikrofone eine Ortung auf wenige Zentimeter genau erfolgen (Abschnitt 6.2). Das ist insofern bemerkenswert, als dass die Kette der Softwarebestandteile aus Zeitsynchronisierung, Soundkartendrift, Signalanalyse und Lampshade-Algorithmus konjunktiv miteinander verbunden und damit sehr sensibel ist: Versagt ein einziges Modul, so ist die ganze Ortung hinfällig.

### 7.1 Herausforderungen

Während die Schallquellen-Ortung mit festen Mikrofonpositionen eine vergleichsweise einfache Aufgabe ist, stellt sie sich mit beweglichen, unbekanntem Mikrofonen deutlich komplizierter dar. Das zeigte sich zum einen an der Erfolgsrate des Algorithmus, zum anderen an der Zahl der notwendigen Iterationen. Im Falle der *bekanntem* Positionen konnte praktisch jedes lösbare Szenario mit nicht kollinearen, und damit uneindeutig positionierten Mikrofonen, eindeutig gelöst werden. Zumindest traten in den praktischen Ortungsanwendungen (Abschnitt 6.2) keine Schwierigkeiten diesbezüglich auf. Die Zahl der notwendigen Iterationen lag mit Standard-Einstellungen in einem Bereich von unter 3.000.

Bei der wechselseitigen Lokalisierung hingegen wurden leicht über 10.000 Iterationen erreicht (Tabelle 4.1) und es war mit der harten Nuss der lokalen Minima zu kämpfen (Abschnitt 4.7), die eine *gültige* Lösung von Szenarien verhinderten. Sie konnten bis zuletzt, außer in Einzelfällen, nicht behoben werden. Damit die wechselseitige Ortung zuverlässig möglich wird, ist es unabdingbar, dass diese Schwierigkeiten beseitigt werden. Eine ausreichende Zahl von Schallsignalen und Mikrofonen mildert das Problem, doch gerade im interessantesten Zahlenbereich von 4 – 5 Mikrofonen, ab dem eine Lokalisierung möglich wird, ist es eklatant mit bis zu 40 % fehlschlagender Ortsbestimmungen – bei idealen Messwerten. Der Anteil könnte bei realistischen, fehlerbehafteten Messwerten noch höher liegen.

Für eine funktionierende Ortung müssen, wie bereits erwähnt, alle Komponenten perfekt zusammenarbeiten. Liefert das Timestamping fehlerbehaftete Messwerte mit einem Bias, so steigt die Wahrscheinlichkeit, dass der Lampshade-Algorithmus unplausible Werte ermittelt. Die komplizierte, wechselseitige Abhängigkeit der Partikel untereinander erhöht diese Wahrscheinlichkeit weiter. Ein Offline-Szenario mit unbekanntem Schallquellen- und Mikrofonpositionen konnte gelöst werden, nachdem die Zeitstempel manuell vorgefiltert worden waren. Bevor diese Szenarien zukünftig unmittelbar und automatisch gelöst werden können, müssen jedoch Filteralgorithmen eingeführt werden, die die Qualität der Timestamps und des Partikelnetzes sicherstellen. Denn während eine Fehlmessung bei bekannten Mikrofonen die Ortung einer einzigen Schallquelle verdirbt, wobei die anderen Schallquellen unangetastet bleiben, führt sie bei unbekanntem Mikrofonen zur Zerstörung des gesamten Netzwerks, da alle Positionen voneinander abhängig sind.

Die vorliegende Arbeit konnte Antworten auf die grundlegende Frage geben, ob die wechselseitige Ortung prinzipiell möglich ist, was dafür notwendig ist, und inwiefern eine reale Ortung mit handelsüblicher Hardware umsetzbar ist. Jedoch können die entwickelten Programme und durchgeführten Experimente nur ein erster Schritt zur Lösung der komplexen Materie darstellen. Zukünftige Entwicklung ist geplant und bereits in die Wege geleitet. Nach Abschluß dieser Masterarbeit soll die Portierung auf mobile Geräte untersucht werden, welche in den kommenden Wochen beginnen wird (Stand: November 2009). Der folgende Abschnitt umreißt einige Eckpunkte einer solchen Portierung.

## 7.2 Portabilität

Im Hinblick auf mögliche Einsatzgebiete der Schallortung bietet sich eine Optimierung nach verschiedenen Kriterien an, um etwa eine Anwendung auf mobilen Geräten zu ermöglichen. Die Portierung von PC zu diesen Systemen führt manchmal zu Erleichterungen, häufig aber zu erschwerenden Bedingungen für die Schallortung.

Natürlich muss ein Prozessor gewählt werden, der von der Rechenleistung und dem Speicherangebot her für die Schallortung geeignet ist. Die Lokalisierung ist rechenintensiv und sehr fließkomma-lastig, weshalb der Prozessor eine Gleitkommaeinheit besitzen sollte. Auf modernen *Netbooks* mit *Atom*<sup>®</sup>-Prozessor stellt das kein Problem dar, auf *Smartphones* hingegen ist nicht zwingend eine Gleitkommaeinheit vorhanden. Der Speicherplatzverbrauch des Algorithmus' ist unproblematisch, aber es sollte möglich sein, die verwendeten Vektor-Libraries einzubinden. Sollte die Lokalisierung grundsätzlich lauffähig sein, so ist sie nicht zeitkritisch. Eine „Progress Bar“ im Benutzerinterface könnte dem Anwender die Wartezeit von einigen Sekunden erleichtern.

Die Aufzeichnung der Schallsignale benötigt eine bestimmte Mindestleistung. Das System muss in der Lage sein, einen Datenstrom von 22.050, besser aber 44.100 Samples

pro Sekunde aufzuzeichnen und nach Timestamps zu durchsuchen. Das Verarbeiten der Samples muss nicht in Echtzeit geschehen, doch der Datenstrom muss mindestens so schnell verarbeitet werden, wie er anfällt. Dazu ist ein geeigneter A/D-Wandler notwendig, wie er in Form von Soundchips in Mobiltelefonen eingebaut ist und ausreichend Prozessorleistung. Kleinere Mikroprozessoren besitzen oft Analogeingänge, diese sind jedoch nicht auf hohe Sampleraten ausgelegt.

Der aktuell verwendete Algorithmus zur Signalverarbeitung ist vergleichsweise sparsam in Bezug auf die benötigte Rechenleistung, er führt lediglich eine RMS-Mittelung und Schwellwertanalysen durch. Wird mittels Fouriertransformation der Frequenzbereich analysiert, etwa zum Vergleich zweier Signalströme untereinander, so steigt die notwendige Rechenleistung stark an. Spezielle DSP-Prozessoren könnten hierbei hilfreich sein.

Am PC führte eine Auslastung des Prozessors durch andere Aktivitäten manchmal zu einer Beeinträchtigung der hardwareseitigen Signalaufzeichnung: Die Pufferposition der Soundkarte verschob sich, was Zeitstempel ungültig machte. Daran trägt das Betriebssystem eine gewisse Teilschuld, welches keine Echtzeitaufzeichnung eines Analogsignals garantieren kann. Auf einem eingebetteten System ohne multitaskingfähiges Betriebssystem kann somit eine Abkehr von PC-Technik sogar von Vorteil sein.

Desweiteren verfügen eingebettete Prozessoren häufig über nicht so exakte Uhren wie Personal Computer. Während der in jedem modernen PC verbaute HPET, oder Multimedia Timer, bei geringer Gangungenauigkeit eine Auflösung im Mikrosekundenbereich liefert, sind Mikroprozessoren häufig nicht so gut ausgestattet. Der Enocean<sup>®</sup> *Dolphin*, eine Chip-Kompilation auf 8051-Basis, verfügt über einen quartzgesteuerten Timer mit 32 kHz, welcher (nach 1:32-Postscaling) eine Auflösung von 1 ms bietet. Für Indoor-Messungen ist diese Auflösung grenzwertig. Der eingebaute SRCO-Timer hingegen löst 10  $\mu$ s auf, muss aber sorgfältig kalibriert werden und läuft alle 0,65 Sekunden über. Desweiteren besitzt der *Dolphin* ein Funkmodul und kann paketweise mit anderen Dolphins kommunizieren. Die Art der Kommunikation mittels des proprietären ERP-Protokolls ähnelt der UDP-Kommunikation eines IP-Netzwerks. Die Samplerate des vorhandenen Analogeingangs ist leider zu gering für eine Soundaufzeichnung. Mit einer Taktfrequenz von 16 MHz und einem Speicherausbau von 32 KB Flash-Speicher für den Code disqualifiziert sich dieser Ultra-Low-Power-Prozessor endgültig für die Schallortung.

Die Lösung liegt in der Mitte: Neue Smartphones wie das *Apple iPhone*<sup>®</sup> 3GS sind in Betracht zu ziehen. Es verwendet den RISC-Prozessor *ARM1176*<sup>®</sup> mit 412 MHz, der mit einem Fließkomma-Coprozessor ausgestattet werden kann [23]. Das iPhone verfügt über ausreichende 256 MB DRAM und eine WLAN-Anbindung nach 802.11g für die Vernetzung der Geräte untereinander oder mit anderen Mobilgeräten [4]. Die Portierung auf das iPhone wird wesentlich dadurch begünstigt, dass die Audio-Library *FMOD*, auf der die Signalaufzeichnung des Timestamping basiert, eine API für das

iPhone besitzt. Dadurch kann eine vollständige Neuentwicklung der sehr systemabhängigen Signalaufzeichnung vermieden werden. Vermutlich müssten nur kleine Anpassungen vorgenommen werden.

Die Verfügbarkeit dieser modernen, leistungsfähigen mobilen Hardware macht die Anwendung der laufzeitbasierten, wechselseitigen Schallortung auf tragbaren Systemen plausibel und realistisch, die positionsfeste Ortung auf Laptops ist sogar bereits Realität. Die angesprochenen Schwierigkeiten könnten durch sorgfältige Weiterentwicklung der bestehenden Ansätze durchaus bewältigt werden und somit könnten eingangs angeführte Szenarien wie mobile Trittschall- oder Gewitterortung in greifbare Nähe rücken.

# Literaturverzeichnis

- [1] Brian Ferris, Dirk Hähnel, and Dieter Fox. Gaussian Processes for Signal Strength-Based Location Estimation. In *Proceedings of Robotics: Science and Systems Conference (RSS)*, 2006.
- [2] Mihail L. Sichitiu and Vaidyanathan Ramadurai. Localization of Wireless Sensor Networks with a Mobile Beacon. In *Proceedings of the First IEEE Conference on Mobile Ad-hoc and Sensor Systems*, pages 174–183, 2004.
- [3] Wikipedia, The Free Enzyklopedia. [http://en.wikipedia.org/wiki/Main\\_Page](http://en.wikipedia.org/wiki/Main_Page). Recv.: May 2009.
- [4] Wikipedia, Die freie Enzyklopädie. <http://de.wikipedia.org/wiki/Hauptseite>. Recv.: May 2009.
- [5] Johannes Wendeberg. Simulation zur Lokalisation von Schallquellen mit variabler Mikrofonanzahl, September 2007. Albert-Ludwigs-Universität Freiburg, Bachelorarbeit.
- [6] Bohnish Banerji and Sidharth Pande. Sound Source Triangulation Game, 2007. Cornell University Ithaca, New York. [http://instruct1.cit.cornell.edu/courses/ee476/FinalProjects/s2007/sp369\\_bb226/sp369\\_bb226/index.htm](http://instruct1.cit.cornell.edu/courses/ee476/FinalProjects/s2007/sp369_bb226/sp369_bb226/index.htm). Recv.: Aug. 2009.
- [7] Jean-Marc Valin, François Michaud, Jean Rouat, and Dominic Létourneau. Robust Sound Source Localization Using a Microphone Array on a Mobile Robot. In *Proceedings International Conference on Intelligent Robots and Systems (IROS)*, pages 1228–1233, 2003.
- [8] Natascha Widder. Berechnung von Mikrofonen durch Schallsignale mit unbekanntem Ort, Oktober 2009. Albert-Ludwigs-Universität Freiburg, Bachelorarbeit.
- [9] Eigen (C++ library for linear algebra) [computer software], 2009. <http://eigen.tuxfamily.org>. Recv.: May 2009.
- [10] Falko Stenzel. Lokalisation von Schallquellen mit variabler Mikrofonanzahl und zufälliger Mikrofonposition, August 2008. Albert-Ludwigs-Universität Freiburg, Studienarbeit.
- [11] Gerd Fischer. *Lineare Algebra*. Vieweg, Braunschweig, 13 edition, July 2002.

- [12] Julian Bader, Benjamin Ummenhofer, and Johannes Wendeberg. HapticBillard – Physikalisch basierte 3D-Billardsimulation mit haptischer Interaktion, April 2009. Albert-Ludwigs-Universität Freiburg, TeamProjekt, <http://sourceforge.net/projects/hapticbillard>. Recv.: June 2009.
- [13] Wolfram Burgard, Cyrill Stachniss, Giorgio Grisetti, Maren Bennewitz, and Christian Plagemann. Introduction to Mobile Robotics, 2008. Albert-Ludwigs-Universität Freiburg, <http://ais.informatik.uni-freiburg.de/teaching/ss08/robotics>. Recv.: June 2009.
- [14] Wikibooks, Die freie Bibliothek. [http://de.wikibooks.org/wiki/Mathematik:\\_Statistik:\\_Glättungsverfahren](http://de.wikibooks.org/wiki/Mathematik:_Statistik:_Glättungsverfahren). Recv.: Sept. 2009.
- [15] David Swaim. Linear Regression, C++ For Scientists and Engineers, 1998. <http://david.swaim.com/cpp/linreg.htm>. Recv.: June 2009.
- [16] Microsoft Help and Support, 2007. <http://support.microsoft.com/kb/939322>. Recv.: May. 2009.
- [17] Zurich University of Applied Sciences: IEEE 1588, 2008. <http://ines.zhaw.ch/en/engineering/ines/ieee-1588/overview.html>. Recv.: May. 2009.
- [18] FMOD, music & sound effects system [computer software], 2009. <http://www.fmod.org>. Recv.: May 2009.
- [19] Michael Tippach. ASIO4ALL – Universal ASIO Driver For WDM Audio [computer software], 2009. <http://www.asio4all.com>. Recv.: Sept. 2009.
- [20] RTAI – the RealTime Application Interface for Linux from DIAPM [computer software], 2009. <https://www.rtai.org>. Recv.: Sept. 2009.
- [21] Waterloo Maple Inc., Maple 9.5 [computer software], 2009. <http://www.maplesoft.com>. Recv.: Oct. 2009.
- [22] Python 2.5 [computer software], 2009. <http://www.python.org>. Recv.: Nov. 2009.
- [23] ARM Ltd. ARM1176JZ(F)-S, 2009. <http://www.arm.com/products/CPUs/ARM1176.html>. Recv.: Sept. 2009.
- [24] Boost C++ Libraries [computer software], April 2009. <http://www.boost.org>. Recv.: June 2009.
- [25] M. Teschner. Simulation der Computergrafik, 2007. Albert-Ludwigs-Universität Freiburg, <http://cg.informatik.uni-freiburg.de/teaching.htm>. Recv.: Jan 2009.