Bachelorarbeit

Simulation zur Lokalisation von Schallquellen mit variabler Mikrofonanzahl

Albert-Ludwigs-Universität Freiburg
Fakultät für Angewandte Wissenschaften
Lehrstuhl für Rechnernetze und Telematik

September 2007
(Rev. June 2011)

Johannes Wendeberg

Erstgutachter: Prof. Dr. Christian Schindelhauer
Zweitgutachter: Prof. Dr. Wolfram Burgard

abstract

Die Lokalisation von Signalquellen stellt einen wichtigen Aspekt der Signaltechnik dar, beispielsweise zur Trittschallortung, Ortung von Gewittern, dem Aufbau eines Sensornetzes. Sie erfolgt gerne unter Verwendung gerichteter Empfänger. Diese sind jedoch kompliziert in der Installation und ihre Position muss geeignet gewählt sein. Eine Alternative stellt die Ortung der Signale mit einer Mehrzahl einfach gestalteter Empfänger anhand der Signallaufzeit dar. Inwieweit lassen sich dadurch Signalpositionen bestimmen? Was ist, wenn die Zahl der Empfänger zu gering ist und die Signalquellen dadurch unterdefiniert? Und welche Aussagen lassen sich treffen, wenn sogar die Position der Empfänger nicht bekannt ist? Diese Bachelorarbeit behandelt eine selbst erstellte Simulation namens „Mics 'n Sounds" zur Schallquellenlokalisation mittels verteilter, zufällig platzierter Mikrofone. Verschiedene Algorithmen wurden implementiert um dies präzise und schnell durchzuführen. Die Ergebnisse werden numerisch und visuell dargestellt, wofür eigens eine 3D-Oberfläche entworfen wurde. Mics 'n Sounds löst nicht nur den konkreten Fall der Lokalisation, falls die Mikrofonpositionenen bekannt sind, sondern kann dank modularer Bauweise auch Grundlage für weiterführende Fragestellungen sein.

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 Schindelhauer vom Lehrstuhl für Rechnernetze und Telematik für die freundliche Betreuung meiner Bachelorarbeit. Außerdem danke ich Andreas Kern und Frederic Schwarz für qualifizierte Ratschläge und Frank Nisch und Michael Rammensee für „technische“ Unterstützung.

Inhaltsverzeichnis

1 Swarmbats

Die Positionsbestimmung von mit Hilfe von Signalquellen und -empfängern ist einer der wichtigsten Bereiche der Signaltechnik. Während es im Bereich der elektromagnetischen Signale bereits eine Vielzahl an Anwendungen gibt (GPS (Kapitel 2.1), RADAR, LORAN, WLAN, uvm.) ist die Anzahl der Systeme, die auf Schallwellen basieren, begrenzt (Cricket [1], SONAR, u.a.).

Bei dem Projekt „Swarmbats“ handelt es sich um einen Ansatz zur Schallquellenortung mit einer Mehrzahl von Mikrofonen. Als Signalgeber kann ein Lautsprecher dienen, oder aber auch „natürliche“ Quellen. Die Position der Mikrofone ist nicht notwendigerweise bekannt, die der Schallquellen sowieso nicht. Inwieweit lassen sich trotzdem relative Positionen wechselseitig bestimmen? Anwendungsgebiete hierfür sind vielfältig. Die erste Idee beinhaltet ein Netz aus Laptops mit eingebauten Mikrofonen. Sie werden in einem Raum verteilt, ihre Position ist zunächst einmal unbestimmt. Untereinander sind sie durch LAN oder WLAN verbunden, um ihre Systemzeit möglichst genau abzugleichen und zum Austausch der empfangenen Daten. Der Zeitabgleich findet über NTP statt. Dieses Protokoll zum Zeitabgleich per Netzwerk ermöglicht über das Internet eine Genauigkeit im Bereich von 10 Millisekunden. Im LAN mit seiner geringeren Latenzzeit sind sogar Genauigkeiten von 0,2 Millisekunden und besser möglich. Ein Schallsignal würde in 0,2 ms eine Strecke von ca. 6,9 cm zurücklegen, was selbst innerhalb eines Raumes genaue Messungen erlaubt. Ein Proband durchquert nun den Raum, vorzugsweise mit Schuhen mit harten Absätzen. Das so entstehene Schallsignal kann aufgezeichnet werden. Falls genug Signale und aufzeichnende Laptops verfügbar sind, so kann über die Laufzeit der Schallsignale ihre relative Position zu den Laptops ermittelt werden. Wenn die Position einer ausreichenden Zahl Laptops absolut bestimmt wurde, so kann sogar die absolute Position des Probanden ermittelt werden. Die Schwierigkeit hierbei ist, dass der Zeitpunkt des Schallsignals nicht bekannt ist. Und das Ergebnis, falls die Positionen unterbestimmt sind (zu wenig Mikrofone) ist auch unklar.

Hierin liegt der Schwerpunkt der Simulation *Mics 'n Sounds*. Sie soll die Fragestellung beantworten, ob und wie die absolute Position von Signalquellen bestimmt werden kann. Gleichzeitig soll sie Unterstützung für die weitere Untersuchung des Projektes bieten[1].

Eine Anwendung, die vom Typ her ähnlich ist, aber sicherlich auch interessant, ist die Ortung von Gewittern. Verschiedene Probanden rufen während eines Gewitters mit ihren Mobilfunkgeräten eine geschaltete Hotline an. Der Donner des Gewitters wird aufgezeichnet. Falls die Handys einen eingebauten GPS-Receiver besitzen und damit ihre Position

[1] Die komplette Lösung des „Swarmbats“-Projektes mit relativen Positionen, Fehlermanagement und Mustererkennung der Signale würde den Rahmen dieser Bachelorarbeit sprengen.

bestimmt ist, so ist die Ortung des Donners einfach. Doch selbst wenn ihre Position unbekannt ist, so ließe sich mit ausreichender Anzahl an Handys und aufgezeichneten Donnerschlägen ihre relative Position zueinander bestimmen.

Eine Signallokalisation mittels Laufzeitanalyse ist nicht die einzige Möglichkeit. Zwei alternative Methoden zur Behandlung der Schallsignale werden nun vorgestellt. Obwohl die Analyse des Frequenzspektrums hier nicht weiter verfolgt wird, wäre sie sicher interessant und könnte in Swarmbats zur Entfernungsbestimmung der Signale implementiert werden. Die Messung der Signalstärke hingegen findet beispielsweise in „Mobile Beacons“ (Kapitel 2.3) Anwendung.

1.1 Breitbandige Schallsignale

Die Lokalisation von Signalen erfolgt entweder durch Laufzeitauswertung zwischen den verschiedenen Empfängern oder durch Analyse der Signaleigenschaften, meistens der Signalstärke. Eine Alternative wäre die Analyse der Frequenz*verteilung*.

Bei Schallsignalen mit weiter Frequenzbandbreite wäre eine Entfernungsbestimmung möglich, falls die Frequenzverteilung in einer bestimmten Entfernung bekannt ist. Dazu muss eine Funktion oder Näherung (z.B. Tabelle) bekannt sein, nach der sich die Frequenzverteilung mit zunehmender Entfernung ändert. Der Vorteil dieser Vorgehensweise ist, dass sie sowohl bei diskreten als auch bei kontinuierlichen Signalen eingesetzt werden kann.

Intuitiv ist einleuchtend, dass ein Schallsignal mit hohen Frequenzen durch die Luftreibung stärker absorbiert wird als mit tieferen. Dies wird auch belegt in [2] („Lokalisation_(Akustik)“):

> Frequenzspektrum: In Luft werden hohe Frequenzen stärker gedämpft als tiefe. Daher wird eine Schallquelle, je weiter sie entfernt ist, desto dumpfer wahrgenommen - die hohen Frequenzanteile fehlen. Für Schall mit bekanntem Spektrum (z. B. Sprache) ist hierüber eine Einschätzung der Entfernung möglich.

Es verschiebt sich die Frequenzverteilung des Signals mit zunehmender Entfernung in Richtung tiefe Frequenzen da die hohen Frequenzbestandteile schneller schwächer werden als die tiefen. Ein Knall oder Donner klingt in großer Entfernung dumpf. Ein Verfahren für die Berechnung des Luftabsorptionskoeffizienten ist in der ISO 9613-1 festgelegt. Die Analyse des Schallsignals erfolgt nach der Zerlegung in Frequenzbestandteile mittels schneller Fouriertransformation.

Eine gewisse Homogenität ist Voraussetzung für das Signal. Die Zusammensetzung der Frequenzbestandteile darf sich nicht mit der Zeit oder bei verschiedenen Signalen ändern. Musik enthält meistens große Dynamik, sowohl in der Lautstärke als auch in der Frequenz. Genau diese Eigenschaften machen Musik interessant, sie ist deswegen aber hier ungeeignet. Menschliche Sprache ist geeignet falls man Methoden hat, die Frequenzverteilung über einen ausreichenden Zeitraum zu mitteln. Hier spielen noch psychologische

Aspekte eine Rolle. Über größere Entfernung könnte ein Mensch dazu neigen, lauter zu sprechen und dabei seine Stimme anzuheben um die Distanz zum Mikrofon zu kompensieren. Dies würde zu einer Verfälschung führen. Eine sorgfältige Kalibrierung ist in jedem Fall notwendig. Gut geeignet ist ein Rauschen, beispielsweise „white noise“, ein zufälliges Signal, welches alle Frequenzbestandteile gleichverteilt enthält.

Versuche hierzu wurden durchgeführt. Ein einfacher Rekorder[2] nahm ein computergeneriertes „white noise“-Signal in variierender Entfernung auf. Während es dem Probanden subjektiv so vorkam, als klinge das Signal in der Nähe schriller und in der Entfernung dumpfer, ließen sich diese Ergebnisse nicht durch die Messungen bestätigen. Eine Frequenzanalyse des Schallsignals im Soundeditor *CoolEdit Pro 2.1* (Hersteller Syntrillium™, gehört inzwischen zu Adobe Systems™) zeigte nicht die erwartete Linearität des Signals. Eine Verschiebung in die Bassbereiche mit zunehmender Entfernung ließ sich zwar erahnen, ging aber unter in der Verzerrung des Signals und der enormen Abnahme der Schallintensität, welche eine weitaus stärkere Rolle spielt. Der Grundgedanke ist wahrscheinlich richtig, aber die verfügbare Messausrüstung war wohl ungeeignet.

Diese Vorgehensweise funktioniert wahrscheinlich nur mit einem Schallsignal. Bei elektromagnetischen Funksignalen ist eine andere Herangehensweise notwendig und zwar das Verhältnis von Entfernung zwischen Signal und Empfänger und der Intensität des Signals. Auch hier ist eine Kalibrierung notwendig.

[2] MP3-Spieler „dnt® MusicTower 20®“

1.2 Signalamplitude

Verschiedene Ansätze zur Lokalisation von Signalquellen verwenden die Veränderung der Signalintensität oder des Signalpegels mit der Entfernung vom Signal, um daraus eine Funktion zur Bestimmung der Distanz abzuleiten.

Die Strahlungsintensität eines elektromagnetischen Signals nimmt mit zunehmender Entfernung quadratisch ab. Es gilt das $(1/r^2)$-Abstandsgesetz für Energiegrößen. Bei doppelter Entfernung ist die Intensität ein Viertel der ursprünglichen Intensität, da sich die Leistung eines punktförmigen Strahlers sinnbildlich auf die Innenseite einer Kugel mit Radius r und der Oberfläche $4\pi r^2$ verteilt. Die Pegeländerung des elektromagnetischen Signals berechnet sich mit $r_2 = 2 \cdot r_1$ zu:

$$\begin{aligned} \Delta L &= 10 \cdot log(\frac{r_1^2}{r_2^2})\, dB \\ &= 20 \cdot log(\frac{r_1}{r_2})\, dB \\ &= 20 \cdot log(\frac{1}{2})\, dB \\ &\approx -6\, dB \end{aligned}$$

Der Schalldruckpegel eines Schallsignals nimmt in zunehmender Entfernung linear ab, mit doppelter Entfernung ist er halb so groß, das entspricht mit $r_2 = 2 \cdot r_1$ der Pegeländerung:

$$\begin{aligned} \Delta L &= 20 \cdot log(\frac{r_1}{r_2})\, dB \\ &= 20 \cdot log(\frac{1}{2})\, dB \\ &\approx -6\, dB \end{aligned}$$

Pegeländerungen können angegeben werden, egal ob es sich bei der Messgröße um eine quadratische oder lineare Größe handelt. So kann man allgemein sagen, dass eine Entfernungsverdopplung, egal ob bei einem Schallsignal oder einem elektromagnetischen, zu einer Pegeländerung von -6 dB führt. Nach vorangegangener Kalibrierung und einem Signal in der Einheit Dezibel kann mit diesem Wissen eine Entfernungsbestimmung durchgeführt werden.

2 Related Work

Um das Problem der Signalquellenlokalisation zu betrachten, war es zunächst wichtig zu sehen, was bereits an Ansätzen zu diesem Thema existiert. Da es eine bedeutende Angelegenheit für viele Bereiche aus Freizeit und Beruf bis hin zur Raumfahrt darstellt, wird die Positionsbestimmung von Signalquellen seit Jahren intensiv untersucht und erforscht. Zwei Gebiete haben sich dabei herauskristallisiert, die Infrastruktur-basierten Systeme und die temporären Ad-hoc-Netze. Erstere verlassen sich auf ein installiertes Netz von Knoten. Es ist enweder die Position der Sender oder die der Empfänger bekannt. Das **G**lobal **P**ositioning **S**ystem (GPS) gehört in diese Kategorie, genau wie das „Sound Source Triangulation Game“, beide werden nachfolgend behandelt. Bei den Ad-hoc-Netzen existieren weniger Anwendungen. Sie haben erst in den letzten Jahren an Bedeutung gewonnen. Das Prinzip ist ein Netz von Knoten, das ohne vorherige Installation seine eigene Infrastruktur „ad hoc“ aufbaut. Die Position der Knoten ist dabei entweder gar nicht bekannt, wird mittels Infrastruktur-Netzen wie GPS ermittelt oder anderweitig vorgegeben. Eine Anwendung zur Distanzmessung, die möglich geworden ist obwohl sie gar nicht in dieser Richtung geplant war, ist WLAN nach IEEE 802.11. Es wird meistens infrastrukturbasiert eingesetzt und verbindet mobile Knoten mit Access-Points. Möglich ist aber auch eine ad-hoc-Vernetzung, wie es viele Treiber für PC anbieten. Und es kann zur Entfernungsmessung zwischen Knoten eingesetzt werden, wie in Kapitel 2.3 beschrieben.

2.1 Global Positioning System

Beim GPS handelt es sich um ein satellitengestütztes System zur Positionsbestimmung zu Land, zu Wasser und in der Luft. Wie in [2] und [3] („Global_Positioning_System“) beschrieben, ist damit heutzutage das NAVSTAR-GPS des US-Verteidigungsministeriums gemeint. Das Global Positioning System ist offiziell seit 1995 in Betrieb und löst das auslaufende System „NNSS“ der US-Marine ab. Nachdem es ursprünglich ausschließlich dem militärischen Bereich vorenthalten war, ist es heute auch für die internationale Öffentlichkeit freigegeben. Anwendung findet es in einer Vielzahl von Bereichen militärischer (z.B. Waffenlenkung, Truppenüberwachung) und ziviler Art, wie Schiffahrt und Luftverkehr. Durch die Freigabe wird es heutzutage hauptsächlich zivil eingesetzt. Es existieren tragbare Empfänger für Outdoor-, Fahrzeug- und Bootsnavigation und viele weitere Anwendungsmöglichkeiten.

GPS basiert auf mindestens 24 Satelliten die sich im Orbit um die Erde in ca. 20 km Höhe befinden. Der Orbit ist nicht geostationär, ein Satellit befindet sich alle 23 Stunden und

55 Minuten an derselben Stelle über der Erde. Alle senden kontinuierlich ihre Position und eine Zeitmarke aus. Ihr Signal lässt sich mit GPS-Empfängern überall auf der Erde empfangen. An jedem Punkt sind jederzeit mindestens vier Satelliten in Reichweite. Für eine erfolgreiche Positionsbestimmung sind theoretisch Signale von drei Satelliten notwendig. Da die meisten Empfänger nicht über die exakte Uhr verfügen, die erforderlich ist, benötigen sie tatsächlich vier Satelliten in Reichweite. So lässt sich die Zeit als letzte Koordinate bestimmen und damit die exakte Position.

Dies wird in *Mics 'n Sounds* nachvollziehbar, welches nach einem ähnlichen Prinzip arbeitet (mit einer anderen Methode). Bei einer Anzahl von drei Mikrofonen kann die Schallquellenposition nur auf einen Kegelschnitt eingegrenzt werden, da der Zeitpunkt der Signalaussendung nicht bekannt ist. Erst mit vier Mikrofonen (die nicht auf einer Ebene liegen) wird die Position eindeutig bestimmbar.

2.2 Sound Source Triangulation Game

Zwei Studenten der Cornell University in Ithaca, New York untersuchten im Frühjahr 2007 einen praktischen Ansatz zur räumlichen Schallortung (s. [4]). Bohnish Banerji and Sidharth Pande schrieben das „Sound Source Triangulation Game“, welches ihr finales Projekt im Seminar „Microcontroller Design“ darstellt.

Ihr Ziel war es, den Zeitpunkt und Ort eines Schallsignals in allen Dimensionen zu berechnen. Der Proband sollte in die Hände klatschen und das System bestimmt die x-, y- und z-Koordinate des Klatschens sowie den Zeitpunkt t. In einer Erweiterung wurde der Proband angewiesen wird, an einem bestimmten Ort zu klatschen. Das System maß, ob er nahe genug am gewünschten Ort in die Hände geklatscht hatte und wurde gegebenenfalls angewiesen, es nochmal zu probieren. Als Benutzerschnittstelle dient ein TV-Monitor der über den Microcontroller angesteuert wird. Vorgabe für das Projekt war, dass es vergleichsweise einfach zu implementieren ist und der Hardwareaufwand günstig bleibt.

Um die 4 Unbekannten zu bestimmen ist ein Setup von 4 Mikrofonen notwendig. Zunächst wurden zur Schallaufzeichnung vier omnidirektionale Mikrofone eingesetzt. Diese wurden später gegen unidirektionale Mikrofone getauscht.

Die Aufgabe, den aufgezeichneten Schallsignalen einen Zeitpunkt zuzuordnen, das „Timestamping“ übernahm ein Atmel Mega32 Mikrocontroller (MCU). Dieser ist mit 16 MHz getaktet und besitzt einen internen Speicher von 2 KB. Er lädt Programmcode und Daten aus 32 KB Flash Speicher bzw. 1 KB EEPROM.

2.2.1 Algorithmus

Die Triangulation wird iterativ nach dem Newtonschen Näherungsverfahren durchgeführt. Dieses Verfahren stellt ein Standardverfahren in der Mathematik dar, um nichtli-

neare Gleichungen numerisch zu lösen. Dabei werden die Nullstellen des Schaubildes ihrer Funktion angenähert. Dies geschieht nach folgendem Prinzip:

- Gegeben ist eine differenzierbare, stetige Funktion f mit der unbekannten Nullstelle n_0 und ein gut gewählter Startwert n_1. Wichtig ist, dass zwischen n_1 und n_0 keine weiteren lokalen Minima oder Maxima von f liegen. Der Fehlerwert d ist die Distanz zwischen $(n_0, 0)$ und $(n_1, 0)$ mit $d = n_1 - n_0$.
- An Punkt $P_1 = (n_1, f(n_1))$ wird die Tangente angelegt mit der Steigung $f'(n_1)$. Sie schneidet die x-Achse im Punkt $(n_2, 0)$, welcher näher an $(n_0, 0)$ liegt als $(n_1, 0)$.
- $P_2 = (n_2, f(n_2))$ dient als neuer Ausgangspunkt für die Tangente.

Das Verfahren wird abgebrochen sobald d kleiner ist als ein gewähltes ϵ. Die Funktion deren Nullstelle bestimmt werden soll, ist

$$f_n(x, y, z, t) = c \cdot (t - t_n) - \sqrt{(x - x_n)^2 + (y - y_n)^2 + (z - z_n)^2} \tag{2.1}$$

mit x, y, z, t als Koordinaten der Schallquelle und x_n, y_n, z_n, t_n von Mikrofon n. Für die Näherung stellten Banerji und Pande die Differenzenquotienten

$$\begin{aligned} f_n(x, y, z, t) \approx f_n(x_0, y_0, z_0, t_0) \quad &+ \quad \left.\frac{df_n}{dx}\right|_{(x_0,y_0,z_0,t_0)} \Delta x \\ &+ \quad \left.\frac{df_n}{dy}\right|_{(x_0,y_0,z_0,t_0)} \Delta y \\ &+ \quad \left.\frac{df_n}{dz}\right|_{(x_0,y_0,z_0,t_0)} \Delta z \\ &+ \quad \left.\frac{df_n}{dt}\right|_{(x_0,y_0,z_0,t_0)} \Delta t \end{aligned}$$

auf, sie führen auf die jacobische Matix:

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

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

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

Iterativ wird der „update vector“ $\Delta\vec{x}$ bestimmt, er enthält die ermittelte, momentane Abweichung vom Zielwert:

$$\Delta\vec{x} = A^{-1}\vec{b} \tag{2.4}$$

Sobald $\Delta\vec{x}$ kleiner ist als $\epsilon = 10^{-6}$, bricht das Verfahren ab.

Die algorithmische Implementation dieser Formeln in die in C geschriebene Software ist trivial und dank des beigelegten und gut kommentierten Quellcodes leicht ersichtlich. Einzig die Ermittlung der inversen Matrix $A-1$ führt zu einigen Schwierigkeiten. Banerji und Pande lösten sie mit Hilfe der Adjunkten von A: Die adjunkte Matrix von A ist die „transponierte Matrix der Cofaktoren“ (s. [2, „Adjunkte“]). Zusammen mit der Determinanten von A lässt sich die Inverse A^{-1} einer invertierbaren n x n-Matrix ermitteln aus $A^{-1} = \frac{Adj(A)}{Det(A)}$.

Leider ist der Algorithmus zu Ermittlung von Adj(A) und Det(A) „hartverdrahtet“, also nicht algorithmisch gelöst sondern tabellarisch: Für jedes Feld A_{ij} werden die korrespondierenden Felder aufgelistet. Damit ist die Methode auf 4 x 4-Matrizen beschränkt und eine Verwendung für beliebige n x n-Matrizen scheidet aus. Für den Versuchsaufbau ist dies freilich nicht notwendig. Die Experimentatoren begründen die Wahl der Adjunkte-Methode gegenüber fortgeschritteneren Verfahren wie dem Gaußschen Eliminationsverfahren damit, dass sie ein vorhersagbares Timingverhalten besitzt weil es keine Sprünge gibt und sie deshalb kompatibler mit der vertikalen Synchronisation der TV-Schnittstelle sei.

2.2.2 Implementation

Der Autor hat den Quellcode für den Näherungsalgorithmus übernommen und soweit adaptiert, dass er in seiner eigenen Software lauffähig war. Da die Software *Mics 'n Sounds* in C++ geschrieben ist, war das einfach zu lösen, da C eine Teilmenge von C++ darstellt. Der Algorithmus funktioniert mit den eigenen Parametern einwandfrei, einzig die Initialisierung der Variablen nur am Programmanfang hat bei Versuchswiederholungen zu Fehlern geführt. Außerdem wurde eine Sicherheitsfunktion eingebaut, die den Algorithmus bei Nichtkonvergenz unterhalb der Epsilongrenze nach einer festgelegten Zahl von Iterationen abbricht, um Endlosschleifen zu vermeiden. Als sinnvoller Startwert für die Näherung hat sich die Mitte des Raumes und des Zeitintervalls erwiesen. Nach 4-6 Iterationen bricht der Algorithmus ab falls er korrekt konvergiert ist. Das Ergebnis ist genauer als 10^{-3} m Deviation von einer möglichen Schallquelle. Durch die Verwendung von `double`-genauen Fließkommavariablen könnte man die Genauigkeit weiter steigern. Da allerdings noch Messfehler eingeführt werden, erscheint dies als nicht notwendig.

Falls die Ortsvektoren der vier Mikrofone nicht linear abhängig sind, so findet der Algorithmus die Schallquelle zuverlässig. Ansonsten bricht er ohne Ergebnis ab und liefert „Hausnummern“ zurück.

Leider lässt sich der Algorithmus von Banerji und Pande nur auf die festgelegte Zahl von 4 Mikrofonen und 1 Schallquelle anwenden. Der mathematische Ansatz ist elegant und

vielversprechend. Neben dem Bayes-Algorithmus und der „Rekursiven Partitionierung“ stellt er den Königsweg zur Lösung des Schallquellenproblems dar.

2.2.3 Hardware

Die anfänglich verwendeten omnidirektionalen Mikrofone schienen für das eindeutige Zuordnen des Signals zuwenig Änderung im Spannungspegel zu erzeugen. Deshalb beschlossen die Experimentatoren, gerichtete Mikrofone einzusetzen bei ansonsten gleichem Aufbau. Hierdurch wurden bessere Resultate erzielt.

Die beschränkten Möglichkeiten des Mikrocontrollers führten dazu, dass das Schallsignal nicht in Software im Mega32 verarbeitet wurde, wie ursprünglich geplant. Da gleichzeitig die Synchronisationssignale für den TV erzeugt werden mussten, hätte diese parallele Belastung das zeitkritische Markieren (= „Timestamping“) der eingehenden Schallsignale behindert. Hinzu kommt, dass erhebliche Fragen aus dem Bereich der Mustererkennung aufkommen um ein Signal als gültiges Eingangssignal von andersartigen Schallsignalen zu unterscheiden (Klopfen statt Klatschen) oder um Schallreflektionen von Wänden zu eliminieren. Da dies den Rahmen des Experiments überstiegen hätte, tat es dem Versuchsaufbau keinen Abbruch, das Schallsignal außerhalb des Microcontrollers zu verarbeiten. Die Detektierung des Klatschens wurde durch einen Schmitt-Trigger realisiert. Ein CMOS-Signal fällt beim Erkennen eines Schallereignisses von 5 Volt auf Masse und wird durch den Trigger nach Abklingen der Schallquelle gehalten. Dies konnte durch die MCU an einem der I/O-Ports gemessen werden und der Zeitpunkt aufgezeichnet werden.

2.2.4 Experiment

Um ihren Algorithmus zu validieren, überprüften Banerji und Pande ihre Berechnungen mit einer Simulation in Matlab. Auch in *Mics 'n Sounds*, wo die Position und der Zeitpunkt der Schallquelle grafisch simuliert werden, lässt sich diese Simulation durchführen.

Die Simulation bestätigte die Erwartungen, zeigte aber die Notwendigkeit, dass sich die Schallquelle an einer zentralen Position zwischen den Mikrofonen befindet. Ansonsten konvergiert die Software gegen ein falsches lokales Minimum. Dies sei aber auch im GPS-System der Fall, so die Experimentatoren.

Leider scheint das System als Ganzes nicht zu funktionieren wie geplant. Die Analyse in Matlab durch die Autoren scheint auf Probleme mit dem Invertieren der jakobischen Matrix A zu führen. Allerdings wird nicht ganz ersichtlich was zu den Schwierigkeiten führt. Obwohl der Algorithmus, der diese Timestamps erstellt, korrekt zu funktionieren schien, lag der Fehler vermutlich bei den Timestamps. Die Messung auf einem Oszilloskop zeigte eine Variation der Messwerte von mehreren Millisekunden. Bei einer Schallgeschwindigkeit von $300\frac{m}{s}$ sind das $30\frac{cm}{ms}$. Fehler dieser Größenordnung kann der Algorithmus nicht auffangen.

Ein Nachbau des Versuches mit *Mics 'n Sounds* zeigte, dass der Algorithmus bei ungünstiger Anordnung der Mikrofone dazu tendiert, zum falschen Nullpunkt zu konvergieren. Als Startwert wurde die geometrische Mitte des Raumes gewählt. Die Messfehlereinstellung lag bei 3% Maximalfehler. Dieser geringe Wert führt bereits zu enormer Abweichung von bis zu etwa einem Drittel der Entfernung zwischen Mikrofon und tatsächlicher Schallquelle. Damit haben sämtliche Algorithmen zu kämpfen, das liegt an der für den engen Raum relativ hohen Geschwindigkeit des Schalls. Aber scheinbar zufällig und schwer nachvollziehbar ermittelte der Newtonsche Näherungsalgorithmus einen vollständig abgelegenen Positionspunkt, da er keine Begrenzung des Raumes und des zeitlichen Messintervalls kennt. Je nach Wahl des Startpunktes konvergiert er sinnvoll oder eben nicht. Dies ist schwer abzuschätzen aufgrund der Vierdimensionalität des Raumes. Ein günstiger Startwert lag oft in der Nähe der Schallquelle. Da diese nicht bekannt ist, wären vorangehende Messungen notwendig. Denkbar ist eine grobe Sondierung des Raumes mit einer der anderen Methoden.

Banerji und Pande schließen zu Recht die verwendeten, langen Kabel als Fehlerursache aus. Die Signalgeschwindigkeit des Stromes liegt um den Faktor 10^6 über der des Schalles und ist daher irrelevant. Letztendlich vermuten sie den Fehler bei der Ladezeit ihrer Kondensatoren die das Timing stören. Ein weiteres Hindernis ist, dass beliebige laute Schallereignisse im Labor die Mikrofone triggern können. Es gibt keine Signalverarbeitung, die den Typ des Schallsignals bestimmt, z.B. anhand von Frequenzeigenschaften oder Amplitudenverläufen. Die Schallerkennung ist somit nicht auf das Klatschen mit den Händen beschränkt.

Durch diese Schwierigkeiten sahen sich die Experimentatoren gezwungen, ihr ursprüngliches Vorhaben umzumodeln. Anstatt der Ortung des frei wählbaren Schallsignals, soll ein Spieler in einem bestimmten Quadranten im Raum klatschen und das System ermittelt, ob es der Richtige war. Es wird nur noch ermittelt, welches Mikrofon als erstes ein Signal erhält und daraus der Quadrant bestimmt. Ein fehlerhaftes Mikrofon reduzierte die Zahl der Quadranten auf drei.

2.2.5 Fazit

Das mehrmals neu gestaltete Spiel arbeitete gemäß seiner Spezifikation. Es konnte ein Klatschen erkennen und einem Quadranten zuordnen. Die Ergebnisse wurden auf der eigens implementierten TV-Schnittstelle ausgegeben. Die Erbauer des Systems zeigten sich aber enttäuscht, dass das System nicht geworden ist wie ursprünglich geplant. Sie hätten zuviel Zeit darin investiert, die omnidirektionalen Mikrofone zum Funktionieren zu bringen und beklagen allgemein Zeitmangel als Ausschlaggeber für das Scheitern.

Dennoch ist ihr System ein einfacher und zugleich effizienter Ansatz um eine Schallortung in der Praxis nachzubauen. Möglicherweise könnte durch eine bessere Wahl der Hardware ein positives Ergebnis erzielt werden. Der Ansatz ist zwar sehr verschieden zu dieser Untersuchung, da es mehr um die technische Durchführbarkeit ging anstatt um theoretische

Simulation und maximale Flexibilität. Eine interessante Plattform zum Experimentieren stellt er dennoch dar.

2.3 Mobile Beacons

Ein Großteil der Überlegungen für basiert auf der Annahme, dass die Schallquelle ortsfest ist und diskrete Signale aussendet. Ihre Position wird durch einen Computer bestimmt, der die Zeitpunkte, zu dem das Signal aufgezeichnet wird, miteinander verrechnet. Eingabedaten sind hierfür ausschließlich Mikrofonpositionen, welche bekannt sein müssen und Empfangszeitpunkte. Eigenschaften des Schallsignals wie Frequenzverteilung und Amplitude werden hierbei nicht betrachtet, ausschließlich der Zeitpunkt des Schallempfangs wurde als Kriterium herangezogen. Eine konkrete Implementation wurde durch die Studenten Bohnish Banerji und Sidharth Pande an der Cornell University vorgeführt, allerdings auf 4 Mikrofone und 1 Schallquelle beschränkt.

2.3.1 Alternative

Zu weiten Teilen ist dieses Prinzip übertragbar auf Funkwellen im RF-Bereich. Auch hier ist eine Ortsbestimmung anhand der Laufzeit des Signals denkbar, wenn auch schwieriger in der Praxis durchzuführen. Aufgrund der hohen Geschwindigkeit der elektromagnetischen Wellen und der damit verbundenen geringen Laufzeitunterschiede sind erheblich schnellere Prozessoren notwendig. Die Schaltung von Banerji und Pande wäre beispielsweise nicht in der Lage, die Laufzeit von RF-Signalen in der Größenordnung von $10^{-8}\frac{s}{m}$ aufzulösen: Das Pollingintervall für die Triggerabfrage („RC time constant“) liegt bei $\approx 5 \cdot 10^{-2}s$.

Effizienter ist die Ausnutzung von Charakteristika der Funksignale, wie ihre mit zunehmender Entfernung abnehmende Signalstärke. So kann ein Empfänger bequem die Entfernung zur Signalquelle bestimmen, falls er eine Funktion findet, die die Entfernung in Abhängigkeit zur Signalstärke angibt. Dies ist bei WLAN nach 802.11 vergleichsweise einfach durchzuführen. Handelsübliche Adapter liefern einen Messwert, der die Signalstärke in dB angibt.

Denkbar ist aber auch der umgekehrte Fall. Nicht die Position der Empfänger ist bekannt, sondern die der Sender. Für viele Anwendungen ist es günstiger, ein Netzwerk aus einer Vielzahl kostengünstiger, einfacher Sensoren zu schaffen, anstatt weniger, aber dafür teurer Geräte.

Sei beispielsweise ein Anwendungsfall der taktischen Geländeaufklärung geplant: Ein Sensornetzwerk wird in ein Testgebiet ausgebracht, um Überwachungsfunktionen zu übernehmen. Es sei notwendig, dass die Position der Sensornodes bekannt ist. Dazu empfangen sie Signale von RF-Sendern, welche auch im Testgelände verteilt sind und deren Position bekannt ist. Die Entfernung zwischen Sender und Empfänger bestimmen sie über eine Funktion der Signalstärke und berechnen ihre Position durch Trilateration. Für eine

erfolgreiche Positionsbestimmung werden in der Ebene mindestens drei Sender benötigt die nicht auf einer Linie liegen dürfen. Die Ergebnisse verbessern sich mit größerer Senderzahl. Damit steigen auch die Kosten. Ein Nachteil ist, dass die Sender zusätzlich im Testareal verteilt werden müssen und ihre Position entweder zeitaufwendig manuell bestimmt werden muss, oder kostenintensiv durch Ausstattung mit technischen Mitteln, wie einem GPS-Empfänger. Dieser kostet ein Vielfaches des RF-Moduls (einfacher Sender für RF-Puls) und wird nutzlos, nachdem der Sender seine Position berechnet hat.

Mihail L. Sichitiu und Vaidyanathan Ramadurai von der North Carolina State University, Raleigh, North Carolina gehen einen anderen Weg. Sie reduzieren die Anzahl der Sender mit bekanntem Ort auf einen einzigen, welcher sich durch das Testgebiet bewegt und hoffen, sogar exaktere Positionsbestimmung zu ermöglichen.

2.3.2 Motivation

Ein Sensornetzwerk kann für eine Menge von Anwendungen verwendet werden, von der genannten taktischen Aufklärung, über die Erkundung und Vermessung unbekannten Gebietes, bis zur Verfolgung von beweglichen Zielen.

> ...the applications of sensor networks are limited only by our imagination. [5]

Sein Verwendungszweck ist stark begrenzt, falls die Position der Sensorknoten unbekannt ist. Deren Ortsbestimmung stellt daher eine der Hauptaufgaben bei der Felderkundung dar. Mit bekannter Position vereinfacht sich beispielsweise die Weiterleitung der Messdaten, falls keine zentrale Instanz existiert, um sie entgegenzunehmen, sondern sie über mehrere Knotenpunkte in eine bestimmte Richtung geroutet werden.

Oftmals ist es schwierig oder unmöglich, die Position der Sensoren manuell zu ermitteln. Sensoren die aus Flugzeugen abgeworfen werden oder sogar aus Mörsern verschossen werden (s. [5]), sind erst einmal unzugänglich. Sie landen in feindlichem und/oder unwegsamem Gebiet und eine alternative Lokalisationsmethode muss angewendet werden. Es bietet sich an, bestehende Infrastrukturen wie GPS zu benutzen. Dies setzt aber teure GPS-Receiver voraus und ihre Anzahl gilt es deshalb zu minimieren.

Sichitiu und Ramadurai stellen eine Methode zur Sensorlokalisation basierend auf dem Bayes Theorem zur Berechnung bedingter Wahrscheinlichkeiten vor (s. auch [6]). Zwei Arten von Nodes werden hierfür vorgestellt: Ein *Beacon* ist ein Node das seine eigene Position kennt, im Experiment ist es ist mit GPS ausgerüstet. Es bewegt sich durch ein Feld von *unknown Nodes*, der eigentlichen Sensoren, und hilft ihnen dabei, ihre Position zu bestimmen. Das Beacon kann ein Mensch sein, der mit einem Sendegerät ausgestattet ist, ein Fahrzeug, aber auch das Flugzeug selbst, aus dem die Sensornodes abgeworfen wurden. Es sendet in regelmäßigen Abständen Signale aus, in welchen die eigene Position enthalten ist. Diese werden von den Nodes empfangen. Über die RSSI[1] können sie die aktuelle Entfernung zum Beacon ermitteln. Auch andere Vorgehensweisen zur Entfernungsmessung sind denkbar. Die Forscher halten auch akustisches Ranging für möglich,

[1] **R**eceived **S**ignal **S**trength **I**ndicator

wie es in [7] vorgestellt wird. Ein Vorteil des RSSI-Ranging ist aber die weite Verbreitung in vielen RF-Receivern, auch das verwendete, handelsübliche WLAN nach IEEE 802.11 besitzt eine RSSI-Ermittlung.

Eine Kommunikation der Nodes untereinander oder mit einer Autorität findet während der Lokalisation nicht statt. Die notwendigen Berechnungen werden auf der Node selbst durchgeführt. Auf diese Weise wird volle Skalierbarkeit des Netzwerks gewährleistet. Ein zentraler Rechner würde bei Vergrößerung des Netzwerkes irgendwann überlastet. Gleichzeitig ist er ein Schwachpunkt für die Zuverlässigkeit des Systems, falls er ausfällt oder nicht erreichbar ist.

2.3.3 Durchführung

Für das Experiment wurden HP iPAQ Pocket PCs mit 802.11b-WLAN-Adaptern ausgestattet. Zunächst musste eine Kalibrierung der Entfernungsmessung durchgeführt werden. Die Signalstärke wurde an mehreren Punkten im Abstand von 2,5 m gemessen und die jeweilige Entfernung aufgeschrieben. Das Ergebnis war eine Funktion der Entfernung, die die Signalstärke (in RSSI) als Rückgabewert hatte. Die Forscher bildeten die inverse Funktion und bestimmten so die Wahrscheinlichkeitsfunktion für die Entfernung bei gegebenem RSSI. Sie ließ sich durch Gaußsche Kurven hinreichend gut beschreiben, so erhielten sie eine stetige Dichtefunktion. Offensichtlich war die Standardabweichung der RSSI-Messungen nicht sehr abhängig von der Entfernung. Bei der inversen Funktion hingegen war das der Fall. Bei einem RSSI von 90 (ca. 5 Meter Entfernung) war sie deutlich geringer als bei 70 (>20 Meter). Für das Experiment heißt das, dass die Messergebnisse präziser werden, wenn das Beacon nahe an der Node vorbeikommt. Das Experiment wurde im Freien auf offener Fläche durchgeführt, so dass keine Hindernisse die Signalübertragung stören. Versuchsweise wurde in etwas bewaldeterem Gebiet gemessen, hierbei stieg die Standardabweichung der Messungen, ansonsten war das Ergebnis vergleichbar.

Nachdem die Sensoren schließlich platziert sind, durchquert das Beacon (montiert auf einem ferngesteuerten Auto) das Testgebiet und sendet Signale mit seiner Position. Eine Node, die dieses Paket empfängt, kann daraus schließen, dass das Beacon irgendwo in der Nähe sein muss und zwar in determinierter Entfernung. Es ergibt sich ein *Constraint* (= „Einschränkung“) C(x, y), dargestellt als Wahrscheinlichkeitsverteilung PDF_{RSSI}[2] in einem Array der Größe (x_{max}, y_{max}). Bei jeder weiteren Messung erhält man eine neue (genauere) Positionsbestimmung NPE(x, y), die sich aus der alten Positionsbestimmung OPE(x, y) und dem neuen Constraint nach dem Satz von Bayes berechnet:

$$P(A|B) = \frac{P(A \cap B)}{P(B)} = \frac{P(B|A) \cdot P(A)}{P(B)} \tag{2.5}$$

Nach einer Messung schränkt sich die eigene Position beispielsweise auf einen Kreis um das Beacon mit festem Radius ein. Nach zwei Messungen (mit dem Beacon an verschiedenen Positionen) bilden sich zwei Peaks (= „Spitze“) heraus. Diese werden schärfer und

[2] **P**robability **D**istribution **F**unction

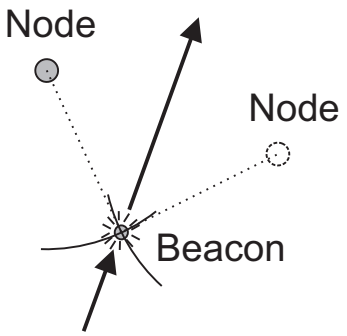

Abbildung 2.1: Eine Node kann nicht erkennen, ob sie links oder rechts des Beacon-Pfades liegt

steiler mit weiteren Messungen. Falls sich die Bewegungsbahn des Beacons linear ist und sich nur auf einer Seite der Node befindet, so ergibt sich das Problem der Symmetrie. Die Node kann nicht beurteilen, auf welcher Seite der Bahn sie liegt, links oder rechts. Theoretisch reichen drei nicht kollineare[3] Messungen aus um die Position eindeutig zu bestimmen. In der Praxis verbessern sich die Ergebnisse (bis zu einem gewissen Grad, siehe „Zusammenfassung“, Abschnitt 2.3.5), wenn das Beacon Kurven fährt, oder die Nodes auf beiden Seiten tangiert. Das wird erschwert weil die Position der *unknown Nodes* eben nicht bekannt ist.

Letztendlich schließt die Bildung des Erwartungswertes $(\check{x}, \check{y})$ die Lokalisation ab. Er wird gebildet über das Array $PosEst(x, y)$ (der Wahrscheinlichkeitsverteilung der Position), indem jeweils $k \cdot PosEst(x, y)$ über das x- und y-Intervall integriert wird, mit $k \in \{x, y\}$.

Die optimale Bahn des Beacons durch das Testgelände wird von Sichitiu und Ramadurai nicht behandelt. Sie schließen die Frage der Bahn ab mit

> Therefore, the beacon trajectory should be designed in such a way that all possible positions are fully covered by at least three non-collinear beacons, and the „grid“ formed by the beacons should be as tight as possible (to increase precision).

Sie fordern also, die Bahn des Beacons solle so eng wie möglich um die Nodes geführt werden. Es gilt eine Abwägung zu treffen. Wenn das Testgelände kurvig und häufig durchquert wird, so wird die Lokalisation genauer, aber um so größer wird auch der zeitliche Aufwand. Betrachtet man diesen Sachverhalt in der Praxis, beispielsweise ein Flugzeug, das als Beacon über feindlichem Gebiet fungiert, so wird man sehen, dass die Gefahr für Besatzung und Gerät dadurch immens ansteigt, nicht nur durch die höhere Zeitdauer des Aufenthalts, sondern auch durch das verdächtige Verhalten.

[3] Drei Messungen „auf einer Linie“

2.3.4 Laufzeit

Eine andere Frage stellt die Art und Weise der Datenrepräsentation. Es ist möglich, die Daten entweder sofort nach dem Empfang an eine (leistungsfähige) Basisstation zu senden, um den nicht unbeträchtlichen, rechnerischen Aufwand abzugeben. Dies sei die einfachste Methode und wird in diesem Ansatz so implementiert. Allerdings beeinträchtigt diese Vorgehensweise die Autarkie des Systems. Deshalb geben Sichitiu und Ramadurai zu bedenken, dass die Berechnung durchweg auf den Unknown Nodes möglich wäre, sogar wenn es sich um „Berkeley Motes“, kleine, kostenreduzierte Sensorplattformen handelt.

Die Berechnung der Wahrscheinlichkeitsverteilung liegt in quadratischer Ordnung $O(n^2)$, da sich die Position der Node in der Ebene befindet. Das Raster besitzt die Größe 100 x 100. Die Berechnungsdauer eines Iterationsschrittes ist auf den betrachteten Plattformen in der Größenordnung von Sekunden. Wenn der Microcontroller nach dem Aussetzen im Zielgebiet einige Minuten mit der Positionsbestimmung beschäftigt ist, und dies nur ein einziges Mal, so halten die Forscher dies für vernünftig. Als Ergebnis bleiben nur die entgültigen $(\check{x}, \check{y})$-Koordinaten und der belegte Speicher von 10 KB für das Grid kann freigegeben werden.

2.3.5 Zusammenfassung

Die Ergebnisse des Experiments entsprechen den Erwartungen, übersteigen sie teilweise sogar. Die Messwerte der Nodes weiter entfernt von der Fahrbahn des Beacons zeigen eine größere Standardabweichung als die näher gelegenen, was zu erwarten war. Doch insgesamt beträgt der Messfehler bei jeder Node (Distanz zwischen gemessener und tatsächlicher Position) weniger als 3 Meter, bei einem Versuchsgelände von über $1200m^2$.

> Considering that we used GPS to measure both the position of the beacon and the positions of the nodes, and that the GPS accuracy was around 3-4m (as reported by the GPS receiver), the accuracy of the algorithm is exceedingly good. Given the GPS accuracy we expected errors on the order of 5-10m. The unexpected accuracy may be due to the differential precision of the GPS receiver, which, in certain situations, can be significantly better than the absolute precision.

Generell tendiert der Messfehler dazu, abzunehmen mit steigender Anzahl von Messpunkten (= empfangenen Positionsmeldungen des Beacons). Weil aber eine schlechte Messung das Gesamtergebnis empfindlich stört, stagniert die Ergebnisqualität mit steigender Zahl der Messungen irgendwann und schwankt um die Größenordnung von 10^0 m, bleibt aber unterhalb von 10^1 m.

Ein Vergleich mit einem Multilaterations-Ansatz wie in [8], welcher dieselben Messdaten erhält, zeigt die Schwächen „normaler“ Multilateration: Die Ergebnisse zeigen massive Ausreißer des Distanzfehlers mit bis zu 25 Metern. Der durchschnittliche Fehler beträgt

über 10 m, anstatt 1,4 m wie beim Bayes-Ansatz. Dem liegt die Anfälligkeit für Messungenauigkeiten bei der Distanzbestimmung zugrunde: Ein einziger schlechter Messwert verfälscht das Gesamtergebnis enorm.

Die hohe Präzision und die einfache Implementierbarkeit sprechen für den Ansatz mit bayesscher, bedingter Wahrscheinlichkeit. Die freie Skalierbarkeit lässt den Versuch mit variabler Nodeanzahl zu, sofern sie in der Lage sind, ihre Position autark zu berechnen und nicht auf einen Basisrechner zurückgreifen müssen. Kritisch bleibt dennoch die Abhängigkeit von einer guten Positionsbahn des Beacons. Falls diese gegeben ist, so wird der Einsatz dieser Implementation mit einem fehlerrobusten, extrem genau lokalisierten Sensornetzwerk belohnt.

3 Mics 'n Sounds

Das Simulationsprogramm *Mics 'n Sounds* ist ein Eigenentwurf im Rahmen der Bachelorarbeit und wurde im Herbst 2007 von Johannes Wendeberg programmiert.

Es dient zur Simulation von Signalquellen im Raum, welche durch entsprechende Empfänger aufgezeichnet werden. Je nach Problemstellung ist die Position der Signalquellen oder der Empfänger unbekannt. Die jeweils Unbekannten gilt es zu lokalisieren. Dafür stehen verschiedene Algorithmen zur Verfügung, sowie eine Infrastruktur, die es erlaubt, relativ einfach eigene Algorithmen zu implementieren. Als Signal ist Schall im hörbaren Bereich ($c = 343\frac{m}{s}$) denkbar, für diesen wurde das Programm konzipiert. Aber nichts spricht dagegen, Funksignale im RF-Bereich zu simulieren ($c = 300.000\frac{km}{s}$). Einige Parameter wie Umweltgröße (momentan 1000 m) und betrachtetes Zeitintervall (momentan 10 sec.) sollten hierfür angepasst werden. Dies lässt sich leicht an zentraler Stelle erledigen.

Sämtliche Software wurde in C++ geschrieben. Zur Entwicklung war diverse Lektüre ([9, 10, 11, 12, 13]) hilfreich. Das Programm basiert auf der Objektbibliothek *Qt* von Trolltech®. Sie bietet sowohl eine weitreichende Palette an grafischen Objekten als auch eine mächtige Unterstützung von Datenstrukturen. Seit einiger Zeit ist der Quellcode als Open-Source unter den Bedingungen der GPL verfügbar. Siehe auch [14].

3.1 Oberfläche

Die Oberfläche ist unterteilt in ein Hauptfenster und ein 3D-Fenster. Das Hauptfenster dient der Verwaltung der Datenelemente, zur Anzeige der Mess- und Rechenergebnisse, zur Kontrolle des Programms und der Algorithmen. Im 3D-Fenster werden die Datenelemente und Messergebnisse visuell ausgegeben. Es ist für das eigentliche Berechnen nicht zwingend notwendig. Die Erfahrung hat aber gezeigt, dass es für das menschliche Vorstellungsvermögen immens hilfreich ist, den dreidimensionalen Raum graphisch darzustellen. Verschiedene Vorgänge wie z.B. die Arbeitsweise des rekursiven Näherungsalgorithmus werden so erst verständlich.

3.1.1 Hauptfenster

Oben im Hauptfenster befinden sich die beiden Item-Fenster für Schallquellen und Mikrofone. Mit dem Kontrollinterface rechts lässt sich beides hinzufügen, bearbeiten oder entfernen. Wenn ein Element markiert ist, so werden rechts seine Koordinaten angezeigt. Diese lassen sich verändern und mit Klick auf „Change“ bestätigen. Bei Schallquellen

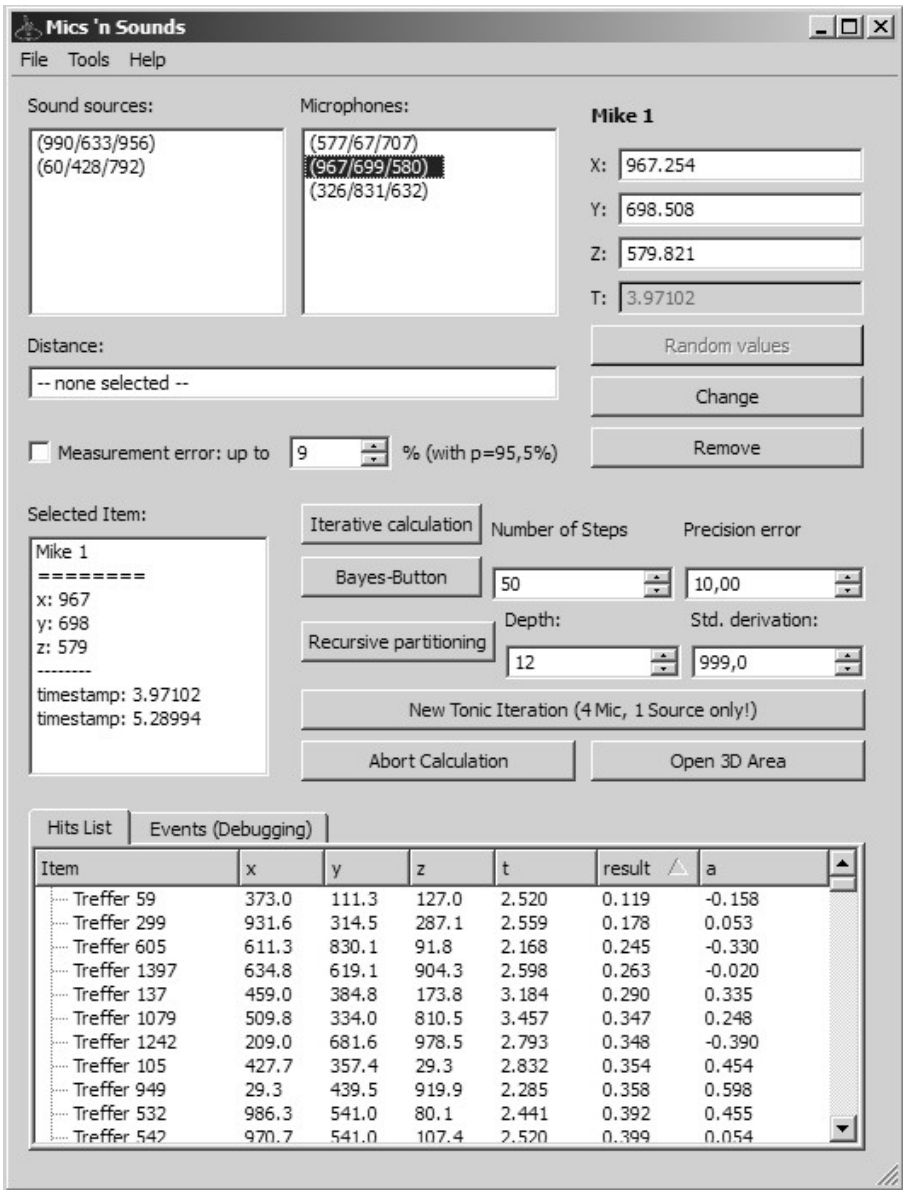

Abbildung 3.1: Beispiel: Hauptfenster mit zwei Schallquellen und drei Mikrofonen. Unten die Ergebnisliste.

kann auch der Zeitpunkt verändert werden. Die Zeitpunkte der Mikrofone hingegen berechnen sich aus der Signallaufzeit der Schallsignale. Die Elementeigenschaften werden auch im Fenster „Selected item“ angezeigt. Unter den Item-Fenstern wird die euklidische Distanz zwischen zwei gewählten Items angezeigt.

Darunter befindet sich die Schaltfläche für den künstlich erzeugten Messfehler. Die Messwerte des Rechners sind bis auf Fließkommagenauigkeit exakt, dies ist in der Natur nicht der Fall. Sämtliche Algorithmen haben mit Messfehlern zu kämpfen und errechnen falsche Werte. Ein probabilistischer Ansatz wie die Bayes-Iteration, die mehrere Messwerte zu einer Aufenthaltswahrscheinlichkeit berechnet, kommt damit besser klar als eine Triangulation, die nur einen einzigen Datensatz verwendet (wie beim „Sound Triangulation Game“). Messfehler liegen in der Natur normalverteilt vor, sie folgen den gaußschen Gesetzen zur Normalverteilung. Deshalb wurde auch der Meßfehleralgorithmus normalverteilt entworfen. Der Zufallszahlengenerator wurde nach der Polarmethode von Marsaglia implementiert. Mehrere Testläufe mit verschiedenen Erwartungswerten und Streuungen bestätigten seine korrekte Implementation. Die Funktion wird mit Erwartungswert E = 0 und Standardabweichung s = 1 ($N[0, 1^2]$) aufgerufen. Der Messfehler wird begrenzt, indem ein prozentuales Intervall angegeben wird, in dessen Bereich er liegt. Da dies bei Normalverteilungen nur probabilistisch möglich ist, liegt er auch nur mit gegebener Wahrscheinlichkeit in diesem Intervall. Momentan ist dieses Intervall auf [-2, 2] der gaußschen Glockenkurve begrenzt in dessen Bereich er mit 95,5%-iger Wahrscheinlichkeit liegt. So liegt beispielsweise ein 8%-iger Fehler im Intervall $[-(t - t_0) \cdot \frac{0.08}{2}, (t - t_0) \cdot \frac{0.08}{2}]$ mit der Ws. 95,5%. $(t - t_0)$ ist die Laufzeit des Schalls. Maximalwerte des Fehlers liegen in der Praxis nicht weit außerhalb davon.

Rechts in der Mitte sind Kontrollelemente für die Algorithmen. Mit ihnen lassen sich Startparameter für die Berechnung der Schallquelle vorgeben und die Algorithmen starten und anhalten. Auf sie wird im Abschnitt 3.1.4 eingegangen. Rechts unten ist ein Karteireiter für das Ausgabefenster für die Trefferliste der Algorithmen („Hits List“) und für Programmereignisse („Events“). Dieses wird hauptsächlich als Debug-Anzeige verwendet. Sämtliche Einstellungen sowie die aktuelle Simulation werden beim Beenden des Programms gespeichert und stehen beim Neustart wieder zur Verfügung.

3.1.2 Menü

Über das Menü „File“ lassen sich Simulationen laden und speichern. Dabei werden alle Simulationsparameter übernommen, insbesondere auch die detaillierten Rechenergebnisse um sie in andere Anwendungen zu exportieren oder statistisch weiterzuverarbeiten. Das Dateiformat ist eine Eigenimplementation von XML namens XMAN 1.0. Es unterstützt die XML-Eigenschaften „Attribut“ in XML-Tags sowie dessen baumartige Hierarchie. Aufgrund der XML-typischen hohen Datenredundanz ist eine Datenkomprimierung mittels ZIP o.ä. bei großen Simulationen ratsam. Aus Gründen der Lesbarkeit der Dateien wurde aber auf eine programminterne Implementation dieser Tools verzichtet.

3.1.3 3D-Oberfläche

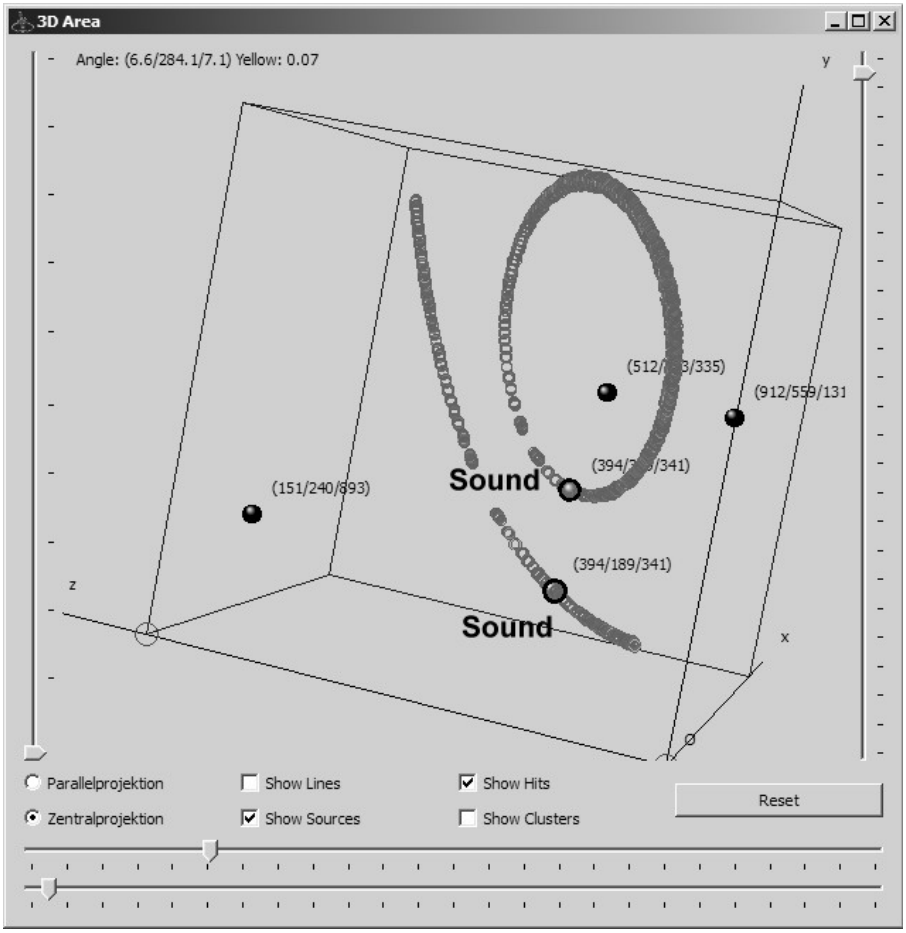

Abbildung 3.2: Beispiel: Drei Mikrofone und zwei Schallquellen. Der rekursive Algorithmus hat die Kegelschnitte identifiziert auf denen sich die Schallquellen befinden.

Über den Button „Open 3D Area“ des Hauptfensters öffnet sich die 3D-Umgebung. Hier werden sämtliche Simulationselemente und die Rechenergebnisse visuell dargestellt. Die Ergebnisse werden nach Treffergüte (*„Deviation“*, also Entfernung von einer möglichen Schallquelle) farblich kodiert. Rot für schlechte Treffer, gelb ist eine Normierung auf eine bestimmte Trefferqualität und grün sind gute Treffer. Mit dem Schieberegler links lässt sich der Wert für die Gelb-Normierung treffen. Sinnvoll ist eine Deviation von 1 als Grenze zwischen guten und schlechten Treffern. Achtung: Die Farbe gibt keinen Aufschluss darüber ob sich tatsächlich eine Schallquelle in der Nähe befindet, dieses kann der Algorithmus prinzipbedingt nicht wissen.

Es stehen zwei Methoden zur räumlichen Interpretation des Gesichtspunktes zur Verfügung. Die Parallelprojektion, welche räumliche Entfernungen zwischen Objekten 1:1 wiedergibt und die Zentralprojektion, welche der räumlichen Verzerrung in zunehmender

Entfernung Rechnung trägt.

- Mit der linken Maustaste kann die 3D-Welt verschoben werden (Insbesondere wichtig, wenn man nah herangezoomt hat).
- Mit der rechten Maustaste dreht sich das Modell um die x- und y-Achse.
- Mit der mittleren Maustaste dreht es sich um die z-Achse.
- Das Scrollrad der Maus zoomt heran und heraus.

Die Drehungen können auch mittels der Schieberegler rechts und unten am Rand vorgenommen werden.

Es hat sich als ein wenig schwierig herausgestellt, das Drahtgittermodell korrekt räumlich zu interpretieren. Manchmal „rutschen“ die hinteren Kanten nach vorne. Eine Drehung des Modells irritiert dann, weil sich der Würfel in die entgegengesetzte Richtung dreht, die man erwarten würde. Es bedarf einiges an Konzentration und etwas Übung, die anderen Kanten wieder geistig nach vorne zu holen. Etwas Unterstützung liefern hierbei die roten Kreise an den Ecken des blauen Würfels. Sie markieren die vorderen, unteren Ecken.

3.1.4 Algorithmen

Momentan sind vier Verfahren zur Berechnung der Schallquellenposition implementiert.

Das Brute-Force-Verfahren ist das primitivste, wenn auch recht robuste Verfahren. Alle Punkte im Raum werden in eingestellter Schrittweite durchlaufen und als Treffer zurückgegeben, falls die Differenz (= „Precision error“) bei jedem Mikrofon kleiner als der im Hauptfenster eingestellte Wert ist. „Number of Steps“ teilt jede Dimension in n Schritte. Die Laufzeit pro Punkt ist also n^4(!). In der Praxis sind Werte von 40 - 50 Schritten und ein Präzisionsfehler von 15 sinnvoll. Es handelt sich beim „Precision error“ nicht um die Deviation, wie sie im Endergebnis berechnet wird. Die Deviation wurde erst nach Implementation des Brute-force-Algorithmus eingeführt.

Das rekursive Verfahren „Recursive Partitioning“ teilt den Raum in vierdimensionale Würfel (s^3 und t), wenn sich eine Schallquelle darin befindet. Dies wird bis zur eingestellten Tiefe durchgeführt. Mit der SpinBox „Deviation“ wird auf Treffer eingegrenzt deren Trefferqualität die eingestellte Standardabweichung nicht übersteigt. Diese Begrenzung ist eigentlich nicht mehr notwendig seit Einführung des „Selektor“. Es empfiehlt sich, mit geringen Rekursionstiefen zu beginnen. Bei klar definierten Simulationen (genügend Mikrofone) ist eine Begrenzung der Standardabweichung nicht notwendig. Auf Gefahren durch die enorme Rekursionstiefe wird im Kapitel 4 detailliert eingegangen.

Nach dem Newtonschen Näherungsverfahren funktioniert die Triangulation wie sie im „Sound Triangulation Game“ verwendet wird. Leider funktioniert sie nur für den Spezialfall 4 Mikrofone und 1 Schallquelle und das auch nur wenn die Mikrofone nicht auf einer Ebene liegen. Dieser Spezialfall wird im Allgemeinen aber sauber und schnell gelöst.

Der Bayes-Algorithmus befindet sich nach wie vor (Stand: September 2007) im Entwicklungsstadium. Es sollte eine Schrittanzahl von 30 pro Dimension nicht überschritten werden.

Auf den programmiertechnischen Unterbau wird im Abschnitt 3.3 eingegangen. Alle Algorithmen werden im Kapitel 4 detailliert beschrieben.

3.2 Datenstrukturen

Die Vielzahl der im Programm verfügbaren Daten und Messwerte erfordert eine gewisse Vorüberlegung bei der Implementierung der verwendeten Datenstrukturen. Die Interaktivität des Programms, die der einfachen Verwendbarkeit der grafischen Benutzeroberfläche Rechnung trägt, verlangt nach einer flexiblen Datenverwaltung. Im Vorfeld sollen Daten aus einer Datei in das Programm eingelesen werden können, um eine Neueingabe bei jedem Programmstart unnötig zu machen. Gleichzeitig soll es aber jederzeit möglich sein, sämtliche Daten im Programm anzupassen, neue Daten hinzuzufügen oder zu entfernen. Während des Ablaufs müssen vorwiegend Listen von Elementen berechnet werden, aber häufig müssen auch einzelne Elemente manipuliert werden, beides soll einfach und bequem machbar sein.

Außerdem wird das Programm so konzipiert, dass ein späterer Anwender den Quellcode leicht übernehmen und verändern kann, ohne sich in die Materie der Datenverwaltung einzuarbeiten. Im Idealfall verwendet er einfach Instanzen der bereits vorhandenen Objektklassen und wendet ihre Methoden an.

Die C-eigene Verwaltung von Datenfeldern, das Array, ist hierfür nur bedingt geeignet: Zunächst hat ein Arrayelement nur einen einzigen Datentyp. Hier wäre problemlos ein Pointer auf eine geeignete Datenstruktur (z.B. für eine Koordinate) möglich. Das Hauptproblem ist aber die komplizierte Handhabung der Elemente. Sämtliche Verhaltensweisen des Arrays beim Hinzufügen, Ändern, Zuweisen von Elementen müssten bedacht und neu programmiert werden.

Eine gute Datenstruktur ist das sprichwörtliche „Rad“ einer Software. Bei der Verwendung von Arrays müsste dieses Rad neu erfunden werden.

Dies ist bereits geschehen in Form einer `QList`. Diese Klasse basiert auf Arrays, stellt aber darüber hinaus noch Methoden bereit, mit Elementen zu arbeiten, beispielsweise `append()` um Elemente hinzuzufügen, ohne Arraygrenzen beachten zu müssen oder `count()` um ihre Anzahl zu bestimmen ohne den Umweg über `sizeof()` gehen zu müssen, oder sie sich irgendwo anders zu merken. Der Datentyp der Listenelemente ist frei wählbar.

Dadurch dass die QList auf Arrays basiert, ist sie schnell beim Hinzufügen und beim Indizieren von nichtsortierten Elementen: Die Komplexität ist $O(1)$. Die meisten anderen Aktionen geschehen in linearer Laufzeit. Eine eventuelle Sortierung in $O(n \cdot log(n))$.

Eine Linked List ist im Vorteil wenn häufig Elemente zu einer sortierten Liste hinzugefügt werden, welches in konstanter Zeit durchgeführt werden kann. Für eine Z-buffer-Darstellung der 3D-Umgebung wurde eine Sortierung der „Nodes“ implementiert, das macht eine Linked List interessant. Allerdings werden die Nodes bei der Drehung des 3D-Objektes in der Praxis einmalig hinzugefügt und später nicht mehr verändert. Dabei ist das unsortierte, sequenzielle Hinzufügen in das Array und anschließende Sortieren zeitlich genauso attraktiv wie das ordnungskonsistente Hinzufügen zu einer Linked List:

Single Linked List: Suchen und sortiertes Einfügen von n Elementen	$O(log(n)) \cdot n = O(n \cdot log(n))$
Array: Sequentielles Einfügen von n Elementen, anschließendes Sortieren (QuickSort, Pivot gut gewählt):	$O(1) \cdot n + O(n \cdot log(n)) = O(n \cdot log(n))$

Das unsortierte Einfügen in die Linked List wird nicht betrachtet, weil dadurch ihr Vorteil, die logarithmische Suche, verspielt wird. Ein weiteres Problem ist die Komplexität der Elemente. Die Sortierung erfolgt anhand der z-Koordinate eines Elementes, die Suche nach Elementen (für das Generieren der Edges) anhand des Elementnamens. Ein Geschwindigkeitsgewinn durch Linked Lists ist deshalb fraglich. Experimente hierzu wurden nicht durchgeführt.

Das Programm besitzt zwei verschiedene Arten von mehrdimensionalen Koordinaten. Zum Einen ein Objekt der Klasse `KoordItem`. Dabei handelt es sich um ein Mikrofon, eine Schallquelle oder einen Treffer in der Berechnung möglicher Schallquellen. Dieses Objekt ist jeweils im dreidimensionalen Raum bestimmt mit einer zusätzlichen Zeitkoordinate oder einer Liste davon. Es besitzt zusätzliche Eigenschaften, wie einen Namen oder ein Listenobjekt, für die grafische Darstellung in einer Liste vom Typ `QListWidget`. Zum Anderen gibt es die Klasse `Node` die für die graphische Repräsentation von 3D-Objekten. Auch sie ist eine dreidimensionale Koordinate. Das ist allerdings die einzige Gemeinsamkeit, ihre Verwendung ist komplett verschieden. Sie ist ein Knotenpunkt in einem 3D-Objekt. Eigenschaften sind neben der x-, y- und z-Koordinate ein Name und ein Knotentyp `„ColorType"`. Der ColorType wird verwendet um unterschiedliche optische Markierungen für die Unterscheidung von Mikrofonen, Schallquellen, Teilen des Raumgitters etc. zu liefern.

Eine `KoordList` ist eine Liste von `KoordItem`-Objekten. Sie enthält Methoden um KoordItems zu verwalten. Sie kann zu einem graphischen Fenster vom Typ `QListWidget` gehören, in dem ihre Elemente angezeigt werden. Wenn ein Objekt zu dieser Liste hinzugefügt wird, wird es im zugehörigen Fenster automatisch angezeigt. Die Angabe dieses Fensters ist aber nicht zwingend notwendig.

Auf ähnliche Weise gibt es die Klasse `NodeList`. Sie besitzt `Node`- und `Edge`-Elemente und Methoden für deren Verwaltung. Die Edges, stellen Verbindungen zwischen Knoten

dar. Eine NodeList entspricht praktisch einem 3D-Objekt welches in der `PaintWidget`-Klasse auf einer Zeichenfläche graphisch dargestellt wird. Die NodeList übernimmt auch die Manipulation des 3D-Objektes, also seine Verschiebung, Skalierung und Drehung.

Edges enthalten Pointer auf zwei Nodes, einen Namen und einen ColorType für die graphische Darstellung. Ihre Nodes wurden komplett als Pointer implementiert. Das hat zwar die Implementation etwas verkompliziert, verbessert aber die Laufzeit. So müssen für eine Neuberechnung des 3D-Objektes nur seine Nodes manipuliert werden, die Edges bleiben unangetastet, obwohl sich auch ihre Position während der Berechnung verändert. Für den Anwender des Programms bleibt die Pointereigenschaft der „Edge-Nodes“ (= Nodes der Edges) verborgen. Er muss nur den Namen zweier Nodes angeben, um die korrekte Pointersetzung kümmert sich die Klasse NodeList. Ein Entfernen einzelner Nodes würde zur Inkonsistenz des 3D-Objektes führen, schließlich müssten auch Edges die diese Node enthalten entfernt werden. Es ist nicht vorgesehen, statt dessen wird stets die komplette Liste neu generiert.

3.3 Programmiertechnik

Das Programm *Mics 'n Sounds* zur Simulation von Signallokalisation ist eine Software, welche sich wohl niemals mit dem Attribut „Final Release“ schmücken kann. Es werden immer Anwendungsfälle existieren, wo die ursprüngliche Konzeption erweitert und neuer Quellcode hinzugefügt werden muss. Ein Ansatz mit neuer Problemstellung und eigenem Algorithmus muss in das Programm eingepflegt werden, dafür müssen sehr wahrscheinlich die vorhandenen Daten neu interpretiert werden. Während der Bachelorarbeit, im Zuge derer das Programm entstand, ergaben sich mehrere Modelle. Allein schon die wenigen Modelle führen zu mehreren Programm-Modi:

- Stetige Schallsignale, Position unbekannt, Analyse erfolgt durch Mustervergleich oder durch Messung der Signalstärke (z.B. RSSI oder Schallvolumen) (nicht implementiert)
- Diskrete Schallsignale, Position unbekannt, Mikrofone mit bekannter Position. Das primäre Modell, Modell zur Gewitterortung und Modell des „Sound Triangulation Game“) (bereits implementiert)
- „Mobile Beacons“ lässt sich zurückführen auf diskrete Signale. Die eine mobile Signalquelle mit diskreten Signalen ist interpretierbar als n Signalquellen an jeder Signalposition zu verschiedenen Zeiten, ein Algorithmus zur Simulation der RSSI wird noch implementiert. (Im Bayes-Ansatz bereits rudimentär enthalten). Vorstellbar wäre auch ein Algorithmus der die Signalquelle tatsächlich bewegt, um beispielsweise Pfadoptimierung zu betreiben.

Die Interpretation der Messdaten ist die nächste Aufgabe. Eine statistische Auswertung wird in dieser Bachelorarbeit nur am Rand betrachtet. Für die implementierten Algorithmen wird sie im Kapitel 4 angesprochen.

Verschiedene Handwerkszeuge auf die sich aufbauen lässt, sind aber gegeben. Zunächst einmal die allgemeine und leistungsfähige Verwaltung der Signalelemente und der sich ergebenden Messdaten mittels `KoordList` und `KoordItem`. Eine Aufteilung der Klassen auf verschiedene Dateien, ermöglicht einfaches Zurechtfinden im umfangreichen Quellcode und verringert die Kompilierzeit, da bei Änderungen in nur einer Datei nicht das komplette Projekt neu kompiliert werden muss. Die komfortable Möglichkeit zur Speicherung dient nicht nur dazu, Simulationen wiederzuverwenden, sondern auch als Exportschnittstelle um Messdaten statistisch weiterzuverarbeiten. Da sie als XML-Baum vorliegen, sind sie mit einem XML-Parser schnell eingelesen. Falls eine andere Anwendung XMAN 1.0 implementiert, so könnten sogar Simulationen importiert werden. Die 3D-Umgebung dient als Anzeige für Daten aller Art: Sofern die Klasse `KoordItem` verwendet wird, muss nur ein neuer `ColorType` definiert werden und die Art und Weise, wie er in der `repaint()`-Methode angezeigt wird. Statische 3D-Objekte werden erzeugt, indem man eine Liste der Nodes und Edges erstellt. Wie das geht, zeigt die Function `addBox()`. Mittels `ColorType` ist ihre Darstellung frei implementierbar. Zuletzt ist noch der Clusteralgorithmus zu nennen. Er fasst eine Liste von Treffern zusammen und versucht sie, einzelnen, möglichen Schallquellen zuzuordnen. Auf dem „k-means“-Algorithmus basierend, wurde er leicht abgewandelt.

1. Dieser bestimmt zunächst eine festgelegte Anzahl zufälliger Clusterzentren.
2. Jedem zu kategorisierenden Item wird das nächstgelegene Clusterzentrum zugewiesen.
3. Das Clusterzentrum wird auf den arithmetischen Mittelpunkt der zugehörigen Items gelegt.
4. Leere Cluster werden gelöscht.

Dieses Verfahren wird ab Schritt (2) wiederholt, bis sich die Zuordnung nicht mehr ändert. Da der k-means aber nicht notwendigerweise konvergiert, und nach einer Ausführung bereits Ergebnisse nahe des Optimums zeigt, wurde auf die Iteration verzichtet. Bei punktuell eindeutig definierten Schallquellen zeigt sich die Stärke des k-means, sie werden zuverlässig identifiziert. Selbst bei linienförmigen Verteilungen von potentiellen Schallquellen im Raum, also Kegelschnitten, erhält man noch sinnvolle Cluster-Einteilungen.

3.3.1 Programmierschnittstellen

Eine Liste der für den Programmierer interessanten Methoden der verschiedenen Klassen befindet sich im Anhang A.1.

Die Funktion copyFrom() verrät eine gewisse Problematik der Edges. Damit ihre Pointer richtig gesetzt werden, muss diese Funktion aufgerufen werden, wenn eine NodeList in eine andere kopiert werden soll. Und das ist notwendig wenn man diesem Schema folgt:

1. NodeList object: Originale Liste, wird nicht verändert

2. NodeList transform: Diese Liste wird gedreht, skaliert usw. dieser Vorgang ist verlustfrei und kann rückgängig gemacht werden
3. NodeList final: Diese Liste wird projiziert, was ein deformierender Vorgang ist. Dies muss vor jedem Anzeigen geschehen.

3.3.2 ColorType

Für jeden Typ von grafischem Objekt der 3D-Oberfläche, sowohl Nodes als auch Edges wird ein ColorType festgelegt, eine frei wählbare Integerzahl. Sie sollte direkt nach dem Erzeugen der KoordList angegeben werden und wird automatisch auf Items und von dort auf Nodes und Edges übertragen. Die Grafikengine in PaintWidget zeichnet anhand dieser Angabe das Objekt.

3.3.3 XML-Engine

Die Engine zum Schreiben von XML-Code ist eine Eigenimplementation und wurde XMAN 1.0 genannt. Sie beherrscht prinzipiell die XML-typische Baumstruktur, wenn diese auch nicht bis in alle Feinheiten ausgereizt wird. Das Einlesen übernimmt der `QXMLSimpleReader`, welches dem Programmierer das Schreiben eines XML-Parsers ersparte. Es musste nur noch ein Content-Handler geschrieben werden der auf eingelesene Tags reagiert. Ein großes Manko ist, dass Anpassungen an KoordItem und KoordList von Hand in den XMLGenerator und den XML-Reader eingepflegt werden müssen. Eine Liste der Members einer Klasse, wie man es aus Java kennt, wäre wünschenswert und könnte über einen Enumerator alle Properties niederschreiben und wieder einlesen. Alte .xm-Dateien werden durch eine Änderung von KoordItem und KoordList möglicherweise inkompatibel!

3.3.4 Threads

Bei komplexen Simulationsumgebungen benötigen die implementierten iterativen und rekursiven Algorithmen eine gewisse Zeit, um zu ihren Versuchsergebnissen zu kommen. Bei der Erhöhung der Ergebnisgenauigkeit steigt die Rechendauer schnell an. Während der Algorithmenteil der Software noch „klassisch“ programmiert war, wurde die gesamte Programmoberfläche blockiert, während der Algorithmus rechnete. Diese Blockade fühlte sich irritierend an und wirkte unprofessionell.

Deshalb wurde entschieden, die Rechenalgorithmen in einem neuen User Thread auszuführen. Sobald eine Berechnung ansteht, wird ein zweiter Thread mit Priorität „Niedriger“ gestartet, was allgemein sinnvoll ist für Hintergrundberechnungen, da sie so die verfügbare CPU-Bandbreite ausreizen, aber andere Threads mit Priorität „Normal“ nicht verlangsamen. Threads sind besser bekannt von der Fähigkeit präemptiver Multitasking-Betriebssysteme, mehrere Prozesse pseudo-gleichzeitig auszuführen, indem sie auf mehrere Kernel Threads verteilt werden. Während eine rechenintensive Software im Hintergrund

arbeitet, ist eine andere Software immer noch bedienbar, wenn auch etwas verzögert, falls der Prozessor komplett ausgelastet ist. Besitzt ein Betriebssystem mehrere Prozessorkerne, so werden diese Prozesse auf die Kerne verteilt und so insgesamt schneller ausgeführt. Auf ähnliche Weise lässt sich ein Programm in User Threads aufspalten. Qt bietet dazu eine Klasse `QThread`, die eine vergleichsweise einfache Erzeugung neuer Threads ermöglicht. Dennoch wird einem die komplexe Verwaltung der Speicherobjekte nicht abgenommen. Lesender und schreibender Zugriff durch die Threads sollte möglichst auf getrennte Objekte erfolgen. Ansonsten müssen gemeinsam genutzte Objekte temporär „gelockt“ und damit der Zugriff auf sie durch andere Threads unterbunden werden. Die Trennung der Objekte ist hier leider nicht möglich, so dass Locking eingesetzt wird. Die Parameter mit denen der Algorithmus aufgerufen wird, werden per „Mutex“ (**Mut**ual **Ex**clusion) geschützt. Das Ergebnis ist beeindruckend: Die Threadprogrammierung ermöglicht eine flüssige Bedienung des Hauptinterface und der 3D-Oberfläche, obwohl eine Berechnung der Schallquellen läuft und die CPU komplett auslastet.

3.4 3D-Transformation

Die Objektbibliothek Qt bietet schon eine Klasse `QGlWidget` für die 3D-Darstellung von Grafik. Ihre Verwendung hätte den Autor interessiert, schließlich verspricht die Verwendung von OpenGL eine optisch sehr ansprechende Darstellung und hardwarebeschleunigte Geschwindigkeit. Die Entscheidung fiel allerdings zugunsten einer Eigenimplementation, um maximale Flexibilität zu gewährleisten. Das Zeichnen der Objekte erfolgt mit Hilfe der `QPainter`-Klasse. Dies ist die Standardklasse zum Erzeugen von Punkten, Strichen und geometrischen Formen auf einer Zeichenfläche vom Typ `QWidget` oder einem Erben davon. Aber auch sie verfügt über komfortable Möglichkeiten, ansehnliche Grafik zu erzeugen, wie Linien-Antialiasing, Farbgradienten und Alpha-Channel-Transparenz.

Die NodeList-Klasse umfasst die üblichen Manipulationsmethoden für 3D-Objekte. Dabei handelt es sich jeweils um eine Matrizenmultiplikation mit dem Koordinatenvektor.

Zur Berechnung werden homogenisierte Koordinaten verwendet: Der Transformationsmatrix M und dem Koordinatenvektor des Punktes (x, y, z) wird eine b-Koordinate hinzugefügt. Sie ermöglicht eine Translation auf die gleiche Weise wie die übrigen Transformationen, anstatt dass sie am Schluss als Addition auf den Koordinatenvektor durchgeführt wird. Ihre Matrix lautet

$$M_t = \begin{bmatrix} 1 & 0 & 0 & dx \\ 0 & 1 & 0 & dy \\ 0 & 0 & 1 & dz \\ 0 & 0 & 0 & 1 \end{bmatrix} (Translation) \tag{3.1}$$

Die Matrizen der übrigen Operationen lauten:

$$M_s = \begin{bmatrix} fx & 0 & 0 & 0 \\ 0 & fy & 0 & 0 \\ 0 & 0 & fz & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} (Skalierung) \tag{3.2}$$

$$M_x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & cos(\beta) & -sin(\beta) & 0 \\ 0 & sin(\beta) & cos(\beta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} (RotateX) \tag{3.3}$$

$$M_y = \begin{bmatrix} cos(\beta) & 0 & sin(\beta) & 0 \\ 0 & 1 & 0 & 0 \\ -sin(\beta) & 0 & cos(\beta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} (RotateY) \tag{3.4}$$

$$M_{rz} = \begin{bmatrix} cos(\beta) & -sin(\beta) & 0 & 0 \\ sin(\beta) & cos(\beta) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} (RotateZ) \tag{3.5}$$

Die entsprechende Matrix M_i wird mit Punktvektor $p = \begin{pmatrix} x \\ y \\ z \\ b \end{pmatrix}$ multipliziert. Sämtliche Transformationen entstammen [15].

Letztendlich wird das 3D-Objekt noch parallel- beziehungsweise zentral-perspektivisch dargestellt. Bei der Parallelprojektion wird dem (x, y)-Teil der Nodes einfach der z-Anteil multipliziert mit einem Korrekturfaktor hinzuaddiert, man erhält $a = a + \delta \cdot z$ mit $a \in \{x, y\}$. Die Zentralprojektion erfolgt durch Einführung eines Betrachtungspunktes vor dem Objekt. Er entspricht einer Kamera oder dem Auge des Beobachters. Durch die Zentralperspektive erhält man einen Fluchtpunkt in unendlicher Entfernung vom Betrachtungspunkt, direkt hinter dem Objekt. Die (x, y)-Anteile der Nodes werden dazu mit z folgendermaßen manipuliert: $a' = f \cdot \frac{a}{f-z}$ mit $a \in \{x, y\}$ und f als Koordinate des Betrachtungspunktes.

4 Algorithmen

4.1 Einleitung

Kernthema dieser Bachelorarbeit stellt die Implementation eines Algorithmus dar, der geeignet ist, die Position möglicher simulierter Schallquellen innerhalb der Umgebung von *Mics 'n Sounds* zu bestimmen. Es schien nicht ohne Weiteres möglich, das Ergebnis kalkulativ zu bestimmen, da hierfür quadratische Gleichungssysteme gelöst werden müssen (s. [16, S. 238ff]). Deshalb versuchen alle Algorithmen, das Ergebnis iterativ anzunähern. Bis auf die Newton'sche Iteration arbeiten alle mit Messpunkten, deren Qualität durch die „Deviation" beschrieben wird.

4.1.1 Deviation

Die *Deviation* eines Messpunktes P wird als Maß eingeführt, welches die Trefferqualität von P an dieser Position bestimmt. Der Punkt wird dafür mit jedem Mikrofon verglichen. Anschließend wird die Standardabweichung der n Zwischenergebnisse berechnet. Dadurch werden im Gegensatz zum Durchschnitt Ausreißer stärker gewichtet: Ein Punkt, an dem für ein Mikrofon ungültige Daten gemessen werden, kann eher keine Schallquelle sein. Formal berechnet sich die Deviation durch

$$\begin{aligned} \text{Diff}_1 &= c \cdot (t_a - \text{Time}(M_1)) - \sqrt{(x_a - x_{M1})^2 + (y_a - y_{M1})^2 + (z_a - z_{M1})^2} \\ &\vdots \qquad \vdots \\ \text{Diff}_n &= c \cdot (t_a - \text{Time}(M_n)) - \sqrt{(x_a - x_{Mn})^2 + (y_a - y_{Mn})^2 + (z_a - z_{Mn})^2} \end{aligned}$$

$$\text{Dev}(P) = \text{stdabw}(\{\text{Diff}_1, ..., \text{Diff}_n\}) \tag{4.1}$$

$Time(M_i)$ ist dabei ein Signalzeitpunkt des Mikrofons M_i. Der Zeitpunkt muss nicht einmal der am besten passende sein, sondern je nach Algorithmus nur unter einem bestimmten Limit liegen. Es ist noch einmal wichtig zu betonen, dass die Deviation nicht die Abweichung von einer tatsächlich existierenden Schallquelle angibt. Diese soll prinzipbedingt unbekannt sein. Die Zwischenergebnisse Diff_i bezeichnen die Entfernung in Metern von einer möglichen Schallquelle bezüglich eines von mehreren Messwerten des Mikrofons M_i, ihre Standardabweichung wird als „Deviation" bezeichnet.

Auf dem Weg zu einem sinnvollen Algorithmus sind mehrere Ansätze entstanden, die verschieden raffiniert versuchen, das richtige Ergebnis anzunähern.

4.2 Brute Force-Ansatz

Eine algorithmische Herangehensweise die sich gleichermaßen als einfach und zuverlässig erwiesen hat, ist der Brute-Force-Ansatz. Bei Brute-Force-Algorithmen wird nicht heuristisch versucht, eine Lösung zu finden, sondern es werden systematisch alle darstellbaren Zustände im Zustandsraum, oder eine nach bestimmten Kriterien gewählte Teilmenge davon, durchprobiert.

Die Fragestellung in diesem Experiment lautet, ob an einem bestimmten Punkt P im Raum zu einem bestimmten Zeitpunkt t ein Schallereignis stattgefunden haben kann. Der betrachtete Punkt ist P_t. Er wird mit allen gespeicherten Signalzeitpunkten a jedes Mikrofons M_i verglichen, $i \in 1, ..., n$. Das Ergebnis Diff_i einer Messung zwischen P zu Zeitpunkt t und Mikrofon M_i zu Zeitpunkt a_i aus der Liste, errechnet sich aus dem Zeitunterschied (= der Laufzeit des Schalls) multipliziert mit der Schallgeschwindigkeit c, minus der euklidischen Entfernung zwischen P und M_i.

$$\text{Diff}_i = c \cdot (t_a - \text{Time}_{Mi}) - \sqrt{(x_a - x_{Mi})^2 + (y_a - y_{Mi})^2 + (z_a - z_{Mi})^2} \tag{4.2}$$

Die Messung gilt als Treffer, falls $abs(\text{Diff}_i) < limit$ ist, einem vorher festgelegten konstanten Wert.

Um zu einem positiven Resultat für Punkt P zu Zeitpunkt t zu kommen, muss sich in der Liste jedes Mikrofones mindestens ein passendes Schallereignis finden lassen. Für jedes Mikrofon muss mindestens ein Treffer gelandet werden. Die Schallereignisse eines einzelnen Mikrofons sind damit disjunktiv („oder“) miteinander verbunden, die Mikrofone miteinander konjunktiv („und“).

Die verschiedenen Punkte P und ihre Zeitpunkte t werden systematisch durchprobiert. Das Universum wird in einer verschachtelten Schleife im Intervall vom Ursprung bis zu jeder Raumgrenze SPACERANGE und an jedem Punkt von Zeitpunkt 0 bis zur Zeitgrenze TIMERANGE durchlaufen. Es ergibt sich ein vierdimensionaler Raum der Komplexitätsordnung $O(n^4)$. Innerhalb jedes Punktes P_t in diesem Raum werden quadratisch viele Berechnungen durchgeführt, eine pro Mikrofon und Empfangssignal, eine Messung liegt also in $O(n^2)$. Der Raum und die Zeit sind nichtdiskret. Im Programm werden sie als Fließkommazahlen einfacher Genauigkeit (Typ float) dargestellt. Die Messungen dagegen erfolgen diskret. Ihre Abweichung vom exakten Treffer wird durch die Differenz Diff_i kompensiert, welche eine gewisse Toleranz erlaubt. Eine wählbare Schrittweite S_s für den Raum und S_t für die Zeit begrenzt die Anzahl der Messungen, sie beträgt $n = (\frac{SPACERANGE}{S_s})^3 \cdot (\frac{TIMERANGE}{S_t})$. Die Anzahl der Vergleiche pro Messung ist $v = |M| \cdot |a|$.

4.2.1 Experiment

Die Messung nur an festgelegten Punkten führt bei niedrig gewählter Toleranz zu überraschenden Ergebnissen. Falls beispielsweise eine „runde“ Schrittweite gewählt wird bei

„runden“ Schallquell- und Mikrofonpositionen, so erhält man eine Ansammlung von Treffern um bestimmte Bereiche, einen Moiré-Effekt. Dieser ist zu vermeiden, indem man die Position von Elementen pseudozufällig wählt.

Zu beachten ist außerdem eine Abwägung der gewählten Schrittzahl. Eine niedrige Schrittzahl kann Treffer übersehen und führt zu einer inhomogenen Trefferwolke. Je höher sie gewählt wird, desto besser sind die Ergebnisse in Genauigkeit und Zuverlässigkeit. Allerdings steigt auch die Ausführungsdauer stark an und zwar exponentiell mit dem Exponenten 4.

Eine sehr hohe Schrittzahl führt gleichzeitig zu einer sehr hohen Trefferrepräsentation derselben Schallquelle. Diese ist bis zu einem gewissen Grad gewollt. Bei einer niedrigen Entfernung um die Schallquelle bildet sich in Kombination mit einer angemessen gewählten Toleranz eine Wolke um die vermutete Schallquelle, die einer Wahrscheinlichkeitsverteilung ähnelt. Eine solche könnte anhand der Daten dieses Experimentes über die Inversfunktion der Deviation angenähert werden. Bei diesem Ansatz hingegen werden die Treffersymbole für die Darstellung der Deviation farblich markiert.

Als brauchbare Parameter haben sich eine Schrittanzahl von 40 und eine Differenz[1] von ca. 10 - 20 herausgestellt. Mit einem handelsüblichen PC der 2-GHz-Klasse lassen sich so ganz brauchbare Ergebnisse erzielen, ohne eine große Wartezeit in Kauf nehmen zu müssen.

4.2.2 Zusammenfassung

Dank seiner Einfachheit und Methode (*„ist hier ein Treffer oder nicht?“*) zeichnet sich dieser Algorithmus durch eine hohe Robustheit aus. Vorhandene Schallquellen werden auch gefunden (außer wenn Schrittweite und Differenz ungünstig gewählt sind). Dabei beschränkt sich der Algorithmus nicht auf einzelne Punkte sondern kann auch Freiheitsgrade richtig erkennen: Eine Schallquelle wird als Linie oder als Fläche wiedergegeben, falls sie unterbestimmt ist.

Ein Manko des Algorithmus ist seine Unfähigkeit, gefundene Schallquellen genauer zu lokalisieren. Eine Erweiterung über die nachgedacht wurde, nennt sich „Position Worms“. Dabei sollten sich Treffer zufällig in verschiedene Richtungen bewegen und beobachten ob diese Verschiebung eine Verbesserung der Treffergüte ergibt. Durch die Einführung der „Rekursiven Partitionierung“ und ihrer hohen Präzision hat sich dies erübrigt.

4.3 Rekursive Partitionierung

Der Brute-Force-Algorithmus ist zuverlässig und kann mit hoher Schrittzahl auch relativ genau werden. Allerdings werden die Ergebnisse nur linear genauer, während der Spei-

[1] Achtung: Beim Einstellbaren Wert im Hauptfenster handelt es sich nicht um die Deviation, sondern um die Differenz an jedem Mikrofon!

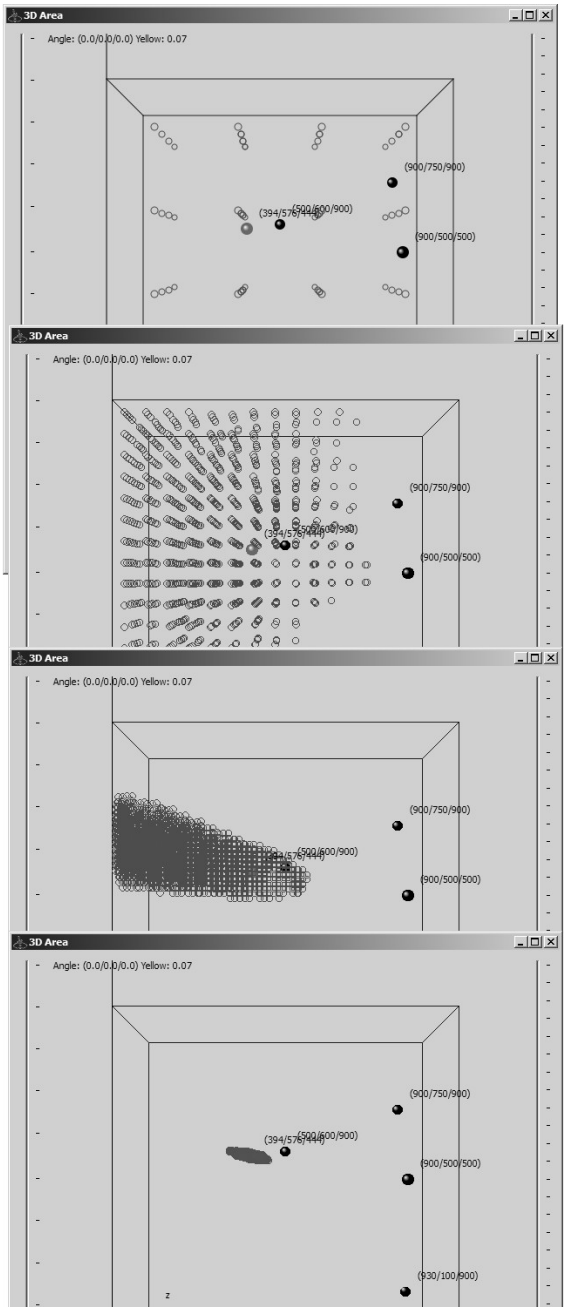

Abbildung 4.1: Der rekursive Algorithmus nähert sich der Schallquelle bei zunehmender Rekursionstiefe weiter an.

cherverbrauch und die Rechenzeit polynomial wachsen. Eine Erhöhung der Schrittzahl von 40 auf 50 verbessert beispielsweise die Deviation rechnerisch um durchschnittlich $\frac{SPACERANGE}{40}$ auf $\frac{SPACERANGE}{50}$, also um den Faktor 1,25. Der Suchraum vergrößert sich aber um den Faktor $(1,25)^4 \approx 2,44$.

Deshalb bestand die Notwendigkeit, einen fähigeren Algorithmus einzuführen. Es stellte sich die Frage, wie man Teile des Raumes in denen sich keine Schallquelle befinden kann, einfach von der Berechnung ausschließen kann. So ließe sich Rechenzeit und Speicher einsparen.

Gegeben sei ein Messpunkt mit Ortsvektor $\vec{a}$. Dieser sei Mittelpunkt des vierdimensionalen Würfels mit den zwei Eckvektoren $\vec{a} - \vec{r}$ und $\vec{a} + \vec{r}$ mit $\vec{r} = (x, y, z, t)$. Von $\vec{a}$ lässt sich die Deviation von einer möglichen Schallquelle $\vec{s}$ berechnen. Sie ist 0, falls $\vec{a}$ sich an der Position von $\vec{s}$ befindet. Die Frage ob sich keine Schallquelle in dem Würfel befindet, lässt sich klären, indem man die maximale, auftretende Deviation innerhalb des Würfels ermittelt. Sie ergibt sich aus der Entfernung, die der Schall in diesem Würfel zurücklegen kann, die maximale Entfernung von $\vec{a}$ in diesem Würfel. Sie beträgt die halbe Strecke von Eckpunkt zu Eckpunkt, was dem Umkreisradius des Würfels entspricht. Diese berechnet sich zu

$$d_{max} = c \cdot (t_a - t_r) + \sqrt{(x_a - x_r)^2 + (y_a - y_r)^2 + (z_a - z_r)^2} \tag{4.3}$$

Falls also $Dev(\vec{a}) > d_{max}$ so liegt mit Sicherheit keine Schallquelle innerhalb des Würfels.

Ob sich eine mögliche Schallquelle innerhalb des Würfels befindet, falls sie nach ihrer Deviation innerhalb des Umkreisradius liegt, lässt sich auf diese Weise nicht klären. Sie kann sich zwischen Würfelgrenze und Umkreisradius befinden. Dieses ist auch nicht notwendig. Der Algorithmus „Rekursive Partitionierung“ arbeitet nun folgendermaßen:

- Als Startwürfel wählt er den gesamten, verfügbaren Raum.
- Falls sich möglicherweise eine Schallquelle darin befindet, dann teilt er diesen Würfel in der Mitte des Intervalls für jede Dimension, also ergeben sich 16 Würfel.
- Für diese wird diese Überprüfung wiederholt.
- Würfel, in denen ausgeschlossen werden kann, dass sich eine Schallquelle darin befindet, werden nicht weiter betrachtet.

So einfach diese Vorgehensweise erscheint, so effizient ist sie gleichzeitig. Die Rechenzeit wird auf ergebnisrelevante Bereiche konzentriert. Dadurch, dass irrelevante Bereiche wieder gelöscht werden, kann Speicher eingespart werden.

4.3.1 Verzweigung

Eine Hürde stellt die gewaltige Anzahl an Rekursionen des Algorithmus dar. Wenn man seinen Suchbaum betrachtet, so besitzt jeder Knoten 16 Verzweigungen. Für ausreichend genaue Ergebnisse ist erfahrungsgemäß etwa die Tiefe 12 notwendig (wird später genauer

betrachtet). Damit ergäbe sich eine Anzahl von $16^{12} \approx 2,8 \cdot 10^{14}$ Blättern. In der Praxis liegt die Rekursionstiefe allerdings deutlich geringer. Falls die Schallquellen eindeutig ermittelbar sind (Mikrofonanzahl ≥ 4), so konvergiert die Zahl $\frac{\text{Treffer bei Tiefe n}}{\text{Treffer bei Tiefe n-1}}$ gegen 1. Ein Beweis dieser Beobachtung steht noch aus. Für linear nicht bestimmbare Schallquellen (3 Mikrofone) nähert sich der Verzweigungsfaktor 2 an. Bei zwei Mikrofonen liegt die Verzweigung etwa bei 4-5. Diese Beobachtungen sind empirischer Natur.

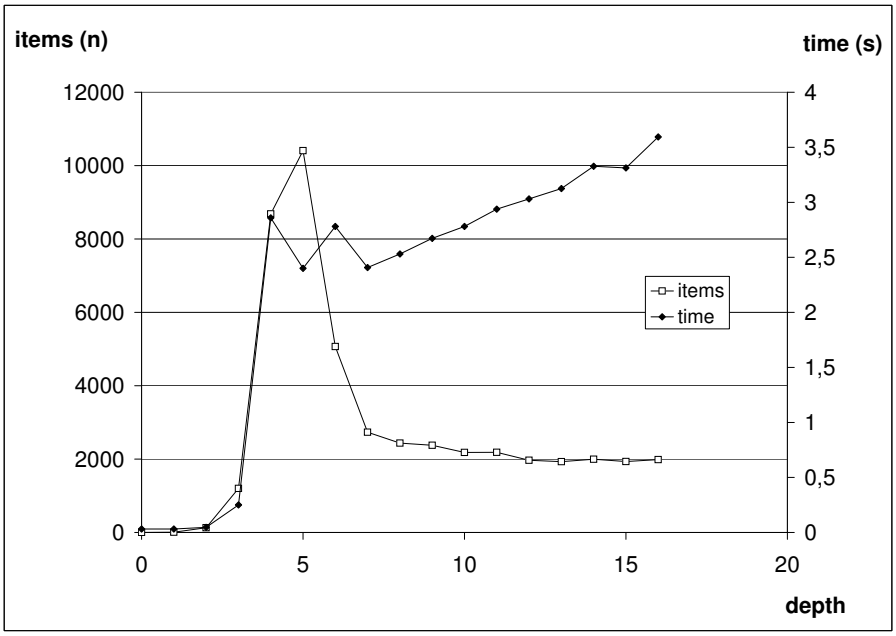

Abbildung 4.2: 10 Schallquellen, 12 Mikrofone zufällig im Raum verteilt, ohne Selektor, man sieht wie der Algorithmus ab Tiefe 6 beginnt zu konvergieren: Der Verzweigungsfaktor nähert sich 1 an, die Trefferzahl steigt nicht weiter an

Falls die Anzahl der Mikrofone gleich 1 ist, sind bezüglich Position und Zeitpunkt des Schallsignals überhaupt keine Aussagen möglich. Der Algorithmus kann keine Würfel verwerfen, der Verzweigungsfaktor ist somit 16. Mit der „Rekursiven Partitionierung“ ist dieser Fall aufgrund der hohen Verzweigung praktisch nicht lösbar, es sei denn man beschränkt sich auf sehr niedrige Rekursionstiefen. Weil er aber uninteressant ist, stört diese Einschränkung nur wenig und wird nicht weiter betrachtet.

4.3.2 Selektor

Die wohl interessantesten Beobachtungen über die Position von Schallquellen lassen sich gewinnen, falls diese im Raum nicht eindeutig bestimmbar sind. Bei drei Mikrofonen ergeben sich Linien, auf welchen die Schallquelle liegen kann, bei zwei Mikrofonen erhält man Oberflächen. Diese sind sehr rechenintensiv und können bei notwendig tiefer Rekursionstiefe zu starker Verzögerung führen. Das Problem liegt weniger in der Zahl der

Rekursionen. Bei drei Mikrofonen ergibt sich für Tiefe 8 eine Zahl von $\approx 10^6$ Treffern welche zwar nur eine kurze Zeit zur Berechnung benötigen. Vielmehr ist die Zahl der Treffer aber weder in der Ergebnisliste noch in der 3D-Umgebung darstellbar.

Die gewählte Lösung ist, die Trefferzahl künstlich zu begrenzen und nur die besten Treffer auszuwählen. Die Berechnung wird zunächst wie gewohnt durchgeführt. Von der Trefferliste werden danach aber nur die besten $m = 3000$ Treffer gewählt und der Rest verworfen. Diese Zahl hat sich als sinnvolle Grenze für die 3D-Darstellung herausgestellt, wo die Umgebung noch flüssig bedienbar ist[2]. Man müsste nun eine Deviationsgrenze d_{lim} einführen, oberhalb der ein Treffer verworfen wird. Nun ist aber über die Treffer nicht bekannt, in welchem Qualitätsbereich sie sich bewegen, wie groß also die minimale und maximale Deviation d_{min} und d_{max} ist, geschweige denn von d_{lim}. Sie zu ermitteln wäre bei 10^6 Treffer zu zeitintensiv. Damit die folgende Methode, „Selektor“ genannt, funktioniert, wird angenommen dass die Deviationen der Treffer linear verteilt vorliegen. Experimente hierzu haben ergeben, dass dies nicht der Fall ist. In den Randbereichen der Verteilung sind gute Treffer besonders gut und weniger gute Treffer besonders schlecht. In der Mitte der Treffermenge verhält sich die Verteilung aber wie erwartet (s. Abb. 4.3). Statt des Testes von sämtlichen Treffern reicht es aus, über eine Sichprobe von 100 Treffern den Durchschnitt d_{avg} und die Standardabweichung d_{stab} der Deviation zu berechnen.

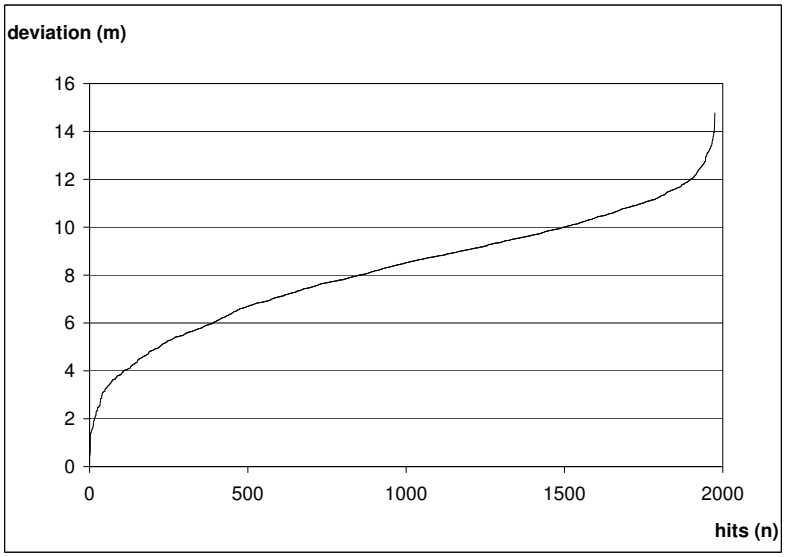

Abbildung 4.3: Beispiel für Deviation-Verteilung der Treffer. Gute Treffer zeigen besonders niedrige Deviation, schlechte besonders hohe. Der Großteil zeigt aber lineare Verteilung.

Das Minimum und Maximum ergibt sich zu

$$d_{max} = d_{avg} + \sqrt{3} \cdot d_{stab} \tag{4.4}$$

$$d_{min} = d_{avg} - \sqrt{3} \cdot d_{stab} \tag{4.5}$$

[2] Auf einem Rechner der 2 GHz-Klasse

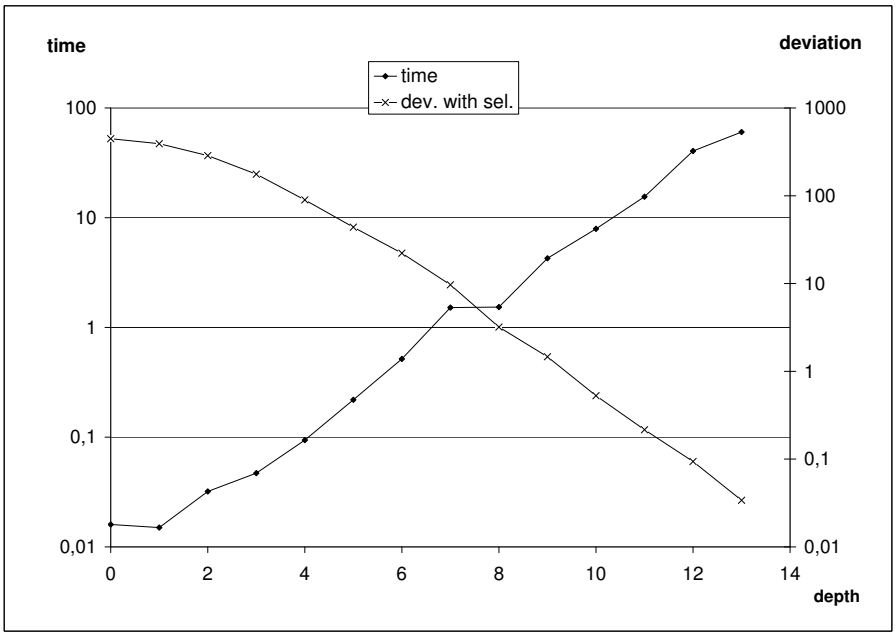

Abbildung 4.4: 1 Schallquelle, 3 Mikrofone. Die Rechendauer vergrößert sich um Faktor 2 mit jeder Iteration, genauso die Trefferqualität.

Bei einer Trefferanzahl n lässt sich das Limit d_{lim} bestimmen, welches zu ca. m Treffern führt:

$$d_{lim} = d_{min} + (d_{max} - d_{min})\frac{m}{n} \tag{4.6}$$

Falls nun eine Trefferzahl jenseits der Grenze d_{lim} festgestellt wird, wird der Algorithmus mit neuer Deviationsgrenze d_{lim} wiederholt. Die sich ergebende Trefferanzahl variiert, da sich d_{lim} in einem extrem engen Bereich bewegt, und es sind maximal 3 Wiederholungen (in Ausnahmefällen 4) notwendig um die richtige Trefferzahl zu erhalten. Leider erhöht sich die Gesamtdauer entsprechend. Eine Ausgabe des Selektor-Algorithmus könnte folgendermaßen aussehen:

```
Divide&conquer calculation started...

Calculation stopped, duration: 25.266 sec.
2526 items found, with average deviation: 0.696

Thread messages:--------------------------
WARNING! Hits count above limit of 3000 items! (146828 items)
Maximum SD is 1.35898, anything above will be cut
(avg is 2.73818, min is 1.30022, max is 4.17614)

WARNING! Hits count above limit of 3000 items! (7865 items)
Maximum SD is 0.930451, anything above will be cut
(avg is 1.04635, min is 0.557572, max is 1.53514)
```

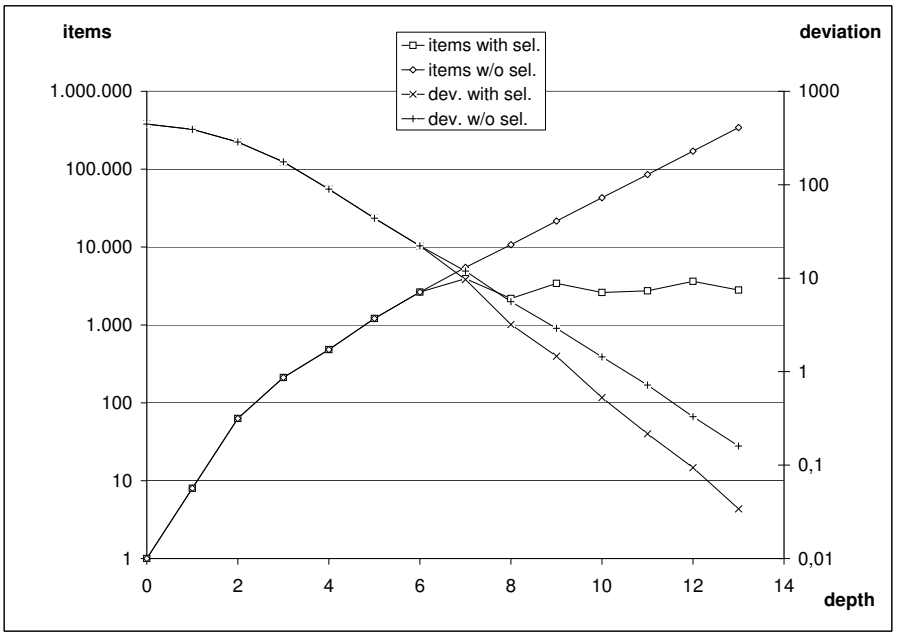

Abbildung 4.5: 1 Schallquelle, 3 Mikrofone. Selektor: Die Zahl der Items stagniert, sobald der Selektor sie begrenzt. Gleichzeitig verbessert sich die Treffergenauigkeit gegenüber der unselektierten Treffermenge.

```
Algorithmus musste 2 mal wiederholt werden
-------------------------------------------
```

Der Vorteil des Selektors wiederum ist, dass sich die Treffergenauigkeit durch die Selektion erhöht (s. Abb. 4.5). Dies ist durch erhöhte Rekursionstiefe aber schneller und effizienter zu erreichen, so dass der Selektor nur eingesetzt wird, wenn es die Trefferzahl tatsächlich erfordert.

Mit der rekursiven Partitionierung ist ein schneller, zuverlässiger Algorithmus entstanden, der nicht zu kompliziert in der Implementation ist. Er findet mögliche Schallquellen umgehend und zuverlässig, wenn sie eindeutig bestimmt sind, ansonsten visualisiert er die Kegelschnitte auf denen sie sich befinden immer noch in akzeptabler Zeit. Durch frei wählbare Verzweigungstiefe ist beliebige Genauigkeit der Treffer erzielbar. Der durch die immense Rekursion auftretenden Blockade bei unterbestimmten Schallquellen kann algorithmisch entgegengewirkt werden.

4.4 Bayes

Der Bayesalgorithmus stellt in dieser Arbeit den aktuellsten Ansatz (September 2007) zur Schallquellenlokalisation dar. Nach Vorlage des in Kapitel 2.3 („Mobile Beacons“) vorgestellten Näherungsverfahrens, wird eine Wahrscheinlichkeitsverteilung der Schallquel-

lenposition erzeugt, welche über die durch die Mikrofone vorgegebenen „Constraints“ (= Bedingungen, Beschränkungen) iterativ verbessert wird.

4.4.1 Constraints

Jedes Mikrofon stellt durch seine Position und durch seine gemessenen Zeitpunkte Bedingungen an die Aufenthaltsorte der Schallquellen: Es ist weder der Ort (x, y, z) noch der Zeitpunkt (t) bekannt, nur der Verlauf der Schallausbreitung. Im vierdimensionalen Raum beschreibt die Position der Schallquelle S einen Punkt auf der Mantelfläche eines Kegels mit dem Mikrofon M als Spitze. Eine Funktion könnte diese Position von S bezüglich M beschreiben, und eine Wahrscheinlichkeitsverteilung C(x, y, z) im Raum (x, y, z, t) erstellen, welche die Position von S bezüglich eines Mikrofones beschränkt. Dies wird für jedes Mikrofon separat durchgeführt.

Ausgangspunkt ist eine Wahrscheinlichkeitsverteilung der vermuteten Schallquellenpositionen. Diese ist am Anfang konstant und gleichverteilt. Aus der gegebenen Wahrscheinlichkeitsverteilung OPE wird dann eine neue Positionsvermutung nach dem Bayes'schen Theorem berechnet:

$$NPE(x,y,z,t) = \frac{OPE(x,y,z,t) \times C(x,y,z,t)}{\int_0^{S_x}\int_0^{S_y}\int_0^{S_z}\int_0^{T} OPE(x,y,z,t) \times C(x,y,z,t)\mathrm{d}x\mathrm{d}y\mathrm{d}z\mathrm{d}t} \tag{4.7}$$

Diese Iteration wird für jedes Mikrofon wiederholt.

Das Ergebnis ist eine Wahrscheinlichkeitsverteilung mit Peaks an jeder vermuteten Schallquellenposition. In „Mobile Beacons“ wird die vermutete Position der Signalquelle über den Erwartungswert der Verteilung gebildet und auf diese Weise der Peak in einen numerischen Wert P(x, y) umgewandelt. Dies ist hier nicht möglich, da im allgemeinen Fall mehrere Schallquellen existieren. Ihre Peaks müssten auf andere Art und Weise identifiziert werden. Die Methode wurde dennoch übernommen und sollte für den Spezialfall genau einer Schallquelle funktionieren.

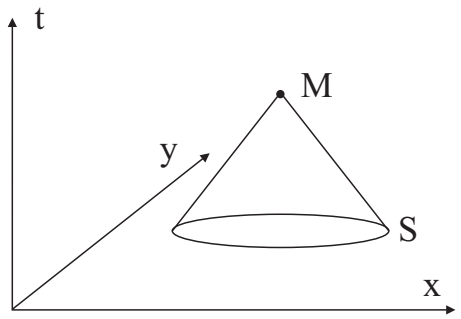

Abbildung 4.6: Skizze der Schallausbreitung mit nur zwei Raumdimensionen (zur Anschaulichkeit): Die Schallquelle liegt auf dem Mantel des Kegels.

4.4.2 Experiment

Die Verrechnung der Constraints mit den Positionsvermutungen und die Bildung des Erwartungswertes scheint korrekt implementiert zu sein. Dennoch funktioniert der Bayes-Ansatz in diesem Experiment nicht wie erwartet. Die ermittelten Positionen tendieren in die richtige Richtung, sind aber sehr ungenau. Grund ist das Fehlen einer geeigneten Funktion zur Berechnung der Constraints selbst. Bisher geschieht die Bewertung eines Punktes in der Wahrscheinlichkeitsverteilung durch Berechnung der Deviation. Durch Invertierung und Normierung auf 1 wird sie verwendbar, um den Constraint zu beschreiben. Sie scheint jedoch nicht zu einer gleichmäßigen Verteilung der Wahrscheinlichkeiten zu führen und entspricht auch nicht der Dichtefunktion dieser Wahrscheinlichkeitsverteilung, da sie diskrete Werte besitzt. Es wäre notwendig, eine stetige Funktion zu generieren die zuverlässigere Werte liefert. In „Mobile Beacons“ bildet sich auf der zweidimensionalen Ebene ein Kraterrand in die dritte Dimension. Die Unknown Nodes liegen für jede Signalquelle[3] auf dem Rand eines Kreises. In diesem Experiment liegen sie hingegen auf dem Mantel eines vierdimensionalen Kegels (s. Abb. 4.6). Die Visualisierung dieses Sachverhaltes stellt sich als nicht-trivial dar. Diese interessante Fragestellung sollte in späteren Forschungsarbeiten unbedingt noch analysiert werden.

Eine weitere Hürde besteht in der Komplexität $O(n^4)$ des Experiments und damit dem Durchlaufen des Universums. Während die Datenmenge in „Mobile Beacons“ mit quadratischer Größe und Auflösung von 100 x 100 im zweidimensionalen Raum übersichtlich bleibt, steigt sie hier enorm an. Die Bildung des Constraints wird ähnlich der Brute-Force-Methode durchgeführt, die Messung erfolgt für jeden Punkt mit vorgegebener Schrittweite. Diese besitzt jedoch mehrere Optimierungen, um die Laufzeit und Datenmenge zu reduzieren. Momentan muss die Berechnung in diesem Ansatz aber für jeden Punkt durchgeführt werden, damit die Feldgröße konstant ist und alle Datenfelder gleich groß. Und so geschieht ein Durchlauf des Universums langsamer als mit der Brute-Force-Methode und zusätzlich muss dieser für jedes Mikrofon durchgeführt werden. Hier besteht Optimierungsbedarf.

4.4.3 Zusammenfassung

Das Bayes-Theorem ist für die betrachtete Fragestellung möglicherweise nicht die beste Methode. Denn mit der „Rekursiven Partitionierung“ wurde bereits eine Methode gefunden die schnell und genau gegen die Ziele iteriert. Sein Einsatz ist vielmehr in der Fehlerverringerung zu suchen. Mehrere Messungen mit einem schnellen Algorithmus, welche allesamt fehlerbehaftet sind, könnten mittels der Bayes-Methode zu zuverlässigeren Ergebnissen führen, als beispielsweise die Bildung des Durchschnitts über alle Messungen.

Der Ansatz erscheint dennoch als vielversprechend, da er sich nahe an die Vorgaben von „Mobile Beacons“ hält: Mihail L. Sichitiu und Vaidyanathan Ramadurai konnten auf

[3] Die Signalquellen und -empfänger sind vertauscht und die Position des Senders ist bekannt, anstatt wie hier die des Empfängers

diese Weise sehr genaue und fehlerimmune Ergebnisse erzielen. Leider steht die Bildung einer brauchbaren Funktion zur Abschätzung der Entfernung momentan noch aus und somit sind die Ergebnisse nur beschränkt brauchbar. Falls eine solche gefunden wird und wirkungsvolle Optimierungen hinsichtlich der Laufzeit erzielt werden, sind bessere Ergebnisse zu erwarten.

4.5 Phantomschallquellen

Wie erwartet kann die Position der Schallquellen durch die vorgestellten Algorithmen eindeutig ermittelt werden, sobald eine ausreichende Anzahl an Mikrofonen verfügbar ist um sie aufzuzeichnen. Interessant ist aber die Frage, was passiert, wenn die Zahl der Mikrofone nicht ausreicht.

Theoretisch müssten vier Mikrofone die Schallquelle in jeder Dimension x, y, z und t bestimmen können, falls sie untereinander nicht linear abhängig platziert sind[4]. Bei einer Schallquelle ist dies auch der Fall. Bei mehreren Schallquellen hingegen treten vereinzelt zusätzliche Trefferclusters auf. Ab einer Anzahl von fünf Mikrofonen ist dies auch der Fall, sie können da aber durch erhöhte Messgenauigkeit (höhere Rekursionstiefe bei „Recursive Partitioning“) eliminiert werden: Fünf Mikrofone und mehr können eine beliebige Anzahl an Schallquellen grundsätzlich eindeutig bestimmen. Warum dies bei vier Mikrofonen nicht möglich ist, konnte nicht geklärt werden. Ob es am Algorithmus liegt, oder prinzipiell am System, ist unklar. Ein Beweis hierzu wäre interessant.

Ein ähnlicher Fall liegt bei unterbestimmten Schallquellen vor: Die Versuche haben ergeben, dass bei zwei und drei Mikrofonen mehr Trefferebenen bzw. Trefferlinien erscheinen, als dass tatsächlich Schallquellen existieren. Ihre Anzahl beträgt bei s Schallquellen und m Mikrofonen maximal s^m (teilweise weniger). Ein Beispiel ist in Abb. 4.7 zu sehen.

Bei großem Zeitunterschied der Signale kann der Algorithmus die Schallquellen unterscheiden: Er ermittelt dann große Messunterschiede der Wahrscheinlichkeit zwischen den Trefferclustern und kann die eine oder andere Position leicht ausschließen. Finden die Schallsignale hingegen gleichzeitig statt oder sehr nahe beinander, so kann keiner der Algorithmen diese unterscheiden. Auch hier stellt sich die Frage, ob das prinzipiell überhaupt möglich ist. Schließlich ist jedem Mikrofon nichts bekannt, außer einer Liste von Timestamps. In dieser Liste ist zwar Eintrag 1 der Schallquelle 1 zuzuordnen, Eintrag 2 der Schallquelle 2 usw. Dieses Wissen wird aber nicht verwendet und darf auch nicht verwendet werden. Diese Zuordnung kann das System nicht wissen. Genau genommen müsste eine Durchmischung der Einträge stattfinden. Darauf kann verzichtet werden, wenn man daran denkt, dieses Wissen nicht zu verwenden. Auf andere Art und Weise müsste ein Algorithmus eine Unterscheidung treffen können der Art: *„Es ist möglich, dass hier eine Schallquelle liegt. Es ist möglich, dass dort eine Schallquelle liegt. Und wenn hier eine liegt dann kann dort keine sein.“* Vermutlich existiert im System aber keine Information,

[4]Davon wird in diesem Abschnitt grundsätzlich ausgegangen.

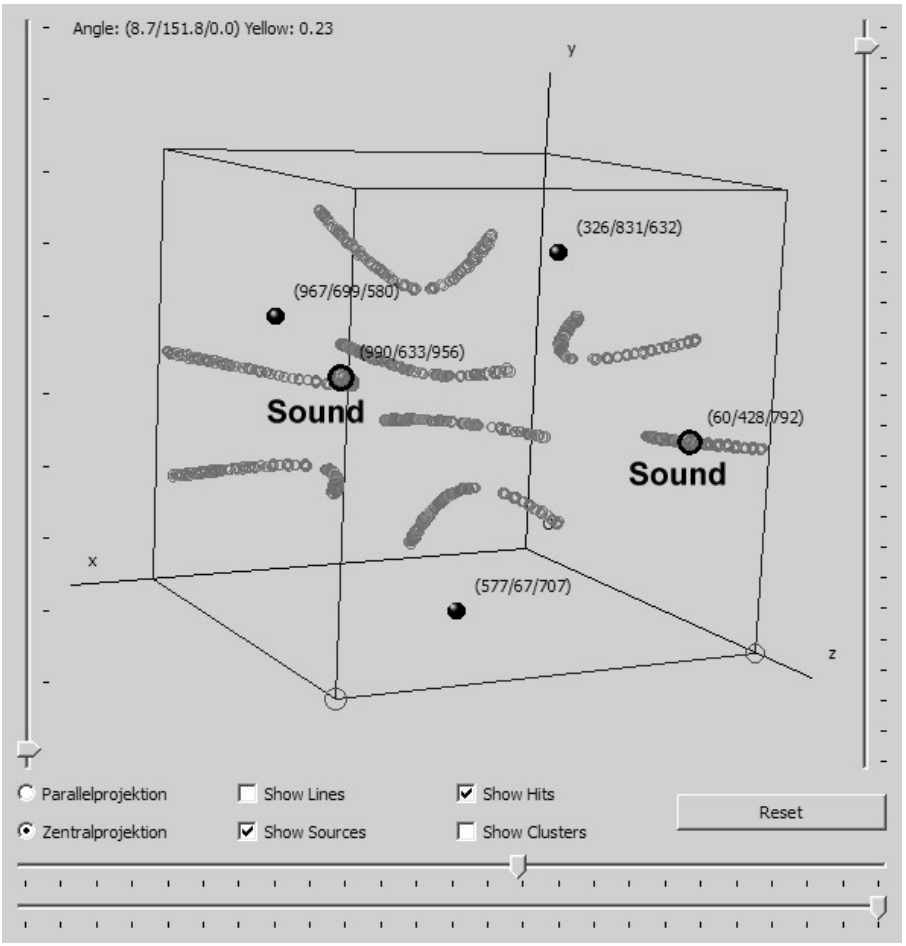

Abbildung 4.7: 3 Mikrofone, 2 Schallquellen. Statt zwei werden 8 Linien gefunden.

mit der sich eine Aussage fällen lässt, die die Existenz der einen Schallquelle bestätigt und gleichzeitigt die der anderen verneint. Auch dies ist noch zu beweisen.

4.6 Ausblick

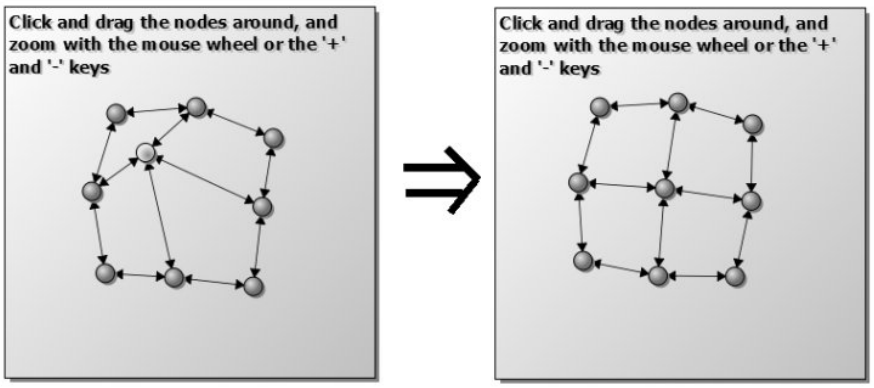

Abbildung 4.8: Qt Example „Elastic Nodes“: Das gummiartige Kräftenetz iteriert gegen einen Zustand in dem die Kräfte zwischen den Knoten minimal sind.

Durch diese Arbeit wurden bereits einige algorithmische Fragestellungen betrachtet und auch gelöst. Ein brauchbarer Algorithmus zur Schallquellenlokalisation ist entstanden. Falls die Positionen der Mikrofone bekannt sind, lassen sich die Positionen der Schallquellen sicher bestimmen, sobald fünf Mikrofone oder mehr zum Einsatz kommen. Die ursprüngliche Frage ging aber noch darüber hinaus und ließ selbst die Position der Mikrofone offen. Falls genügend Mikrofone und Schallquellen zur Verfügung stehen, könnte eine komplette Analyse der Positionen relativ zueinander stattfinden, ohne dass eine einzige Position absolut bekannt ist. Die Mikrofone müssten die Schallquellen ermitteln und solange wechselseitig iterieren bis ein komplettes Netz der Positionen zur Verfügung steht. Dafür könnte ein „Kräftenetz“ zum Einsatz kommen, ein gummiartiges Netz, welches versucht, Spannungen zwischen Knoten zu neutralisieren. Ein einfaches Beispiel ist in Abbildung 4.8 zu sehen. Der Algorithmus sollte auch in der Lage sein, unterdefinierte Netze zu bestimmen: Das sind Knoten im Kräftenetz, die nicht mit Gummibändern verbunden sind sondern frei verschiebbar sind. Auch das Gegenteil ist zu analysieren: Netze in denen sich ein Kräfteausgleich nicht herstellen lässt, weil Knoten widersprüchlich zueinander überbestimmt sind. *Mics 'n Sounds* könnte hierzu als Simulationsgrundlage verwendet werden, in die weitere Berechnungsverfahren implementiert werden.

Literaturverzeichnis

[1] N. Priyantha, A. Chakraborthy, and H. Balakrishnan. The Cricket Location-Support System. In *Proceedings of International Conference on Mobile Computing and Networking*, pages 32–43, 2000.

[2] Wikipedia, Die Freie Enzyklopädie, 2007. `http://de.wikipedia.org/wiki/Hauptseite`. Recv.: 2007.

[3] Wikipedia, The Free Enzyklopedia, 2007. `http://en.wikipedia.org/wiki/Main_Page`. Recv.: 2007.

[4] Bohnish Banerji and Sidharth Pande. Sound Source Triangulation Game, 2007. `http://instruct1.cit.cornell.edu/courses/ee476/FinalProjects/s2007/sp369_bb226/sp369_bb226/index.htm`. Recv.: 2007.

[5] 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.

[6] Christian Schindelhauer. Wireless Sensor Networks, 17th Lecture, 2007. `http://cone.informatik.uni-freiburg.de/teaching/lecture/wsn-w06/index.html`. Recv.: 2007.

[7] L. Girod and D. Estrin. Robust range estimation using acoustic and multimodal sensing. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2001)*, 2001.

[8] A. Savvides, C. C. Han, and M. B. Srivastava. Dynamic fine-grained localization in ad-hoc networks of sensors. In *Proc. of Mobicom 2001*, pages 166–179, 2001.

[9] Arnold Willemer. *Einstieg in C++ : professioneller Einstieg in die Programmierung.* Galileo-Press, 2nd edition, 2005.

[10] Bjarne Stroustrup. *Design und Entwicklung von C++.* Addison-Wesley, 1st edition, 1994.

[11] Bjarne Stroustrup. *Die C++-Programmiersprache.* Addison-Wesley, 4th edition, 2000.

[12] André Willms. *Einstieg in Visual C++ 2005.* Galileo-Press, 1st edition, 2007.

[13] Andrew Koenig and Barbara E. Moo. *Accelerated C++, Practical Programming by Example.* Addison-Wesley, 2nd edition, 2000.

[14] Qt Reference Documentation (Open Source Edition), 2007. http://doc.trolltech.com/4.3/index.html. Recv.: 2007.

[15] Tobias Gross, Nils Faltin, Roman Mülchen, and Michael Plath. Grafiti, 2001. http://olli.informatik.uni-oldenburg.de/Grafiti3/grafiti/flow1/page1.html. Recv.: 2007.

[16] Holger Karl and Andreas Willig. *Protocols and Architectures for Wireless Sensor Networks*. Wiley, 1st edition, 2006.

[17] Tobias Erbsland and Andreas Nitsch. Diplomarbeit mit LATEX, 2002. http://drzoom.ch/project/dml. Recv.: 2007.

[18] Stefan Buchholz. Evaluation von Remote-Angriffsmethodiken auf vernetzte IT-Systeme, 2006. University of Freiburg, bachelor thesis.

[19] Lance Doherty, Kristofer S. J. Pister, and Laurent El Ghaoui. Convex position estimation in wireless sensor networks. In *INFOCOM 2001. Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 3, pages 1655–1663, 2001.

[20] Nate Kohl. C/C++ Reference, 2007. http://www.cppreference.com. Recv.: 2007.

[21] Juan Soulie. C++ Language Tutorial, 2006. http://www.cplusplus.com/doc/tutorial. Recv.: 2007.

[22] Jürgen Weinelt. Der LATEX-Index, 2007. http://www.weinelt.de/latex. Recv.: 2007.

[23] MiKTeX 2.6 Manual, 2007. http://docs.miktex.org/manual. Recv.: 2007.

A Anhang

A.1 Liste der Programmierschnittstellen

Zwar existieren in Mics 'n Sounds noch mehr Funktionen in den implementierten Klassen. Es ist aber sinnvoll, nur die Schnittstellen anzugeben, auf die „von außen“ zugegriffen wird. Nur sie sind für andere Programmierer, welche die Software erweitern, interessant.

A.1.1 KoordItem

Dieses Element stellt ein Signal, einen Empfänger oder einen (durch einen Algorithmus erzeugten) Treffer dar. Es besitzt die räumlichen Koordinaten `x, y und z`, zwei Listen `t` und `a`, einen `ColorType` und ein `result`. In `t` stehen die Timestamps des Items, das ist 1 bei Signalquellen und Treffern, und n (= Anzahl der Signalquellen) bei Mikrofonen. In `a` stehen die Messergebnisse der Treffer. Sie werden in `result` zusammengefasst.

`double x;`
`double y;`
`double z;`
`double result;`
`double colortype;`
`QList<double> t;`
`QList<double> a;`
	Members von `KoordList`.

`KoordItem();` `KoordItem(double x, double y, double z);`	Erzeugt ein neues Element, wahlweise mit Parametern x, y, und z.
`QString text();`	Liefert eine Bezeichnung zurück.
`QString det_text();`	Liefert eine detaillierte Bezeichnung zurück.

`void addToListWidget(`
`QListWidget *par);`
`void removeFromListWidget();`
	Fügt Item zu einem QListWidget hinzu oder entfernt es. Wird nur von KoordList verwendet.

`float distance(KoordItem k);`	Gibt die euklidische Distanz zwischen zwei KoordItems an.

A.1.2 KoordList

In dieser Klasse werden KoordItems gespeichert und verwaltet.

`QString type;`	Typ zur Identifikation innerhalb des Programms und in der XML-Datei. Sollte nach Instanzerzeugung zugewiesen werden.
`int colortype;`	ColorType, sollte nach Instanzerzeugung zugewiesen werden.
`QList<KoordItem> items;` `KoordItem operator[](int i);`	Liste der KoordItems. Rückgabe eines Items entweder durch `s.items[int i];` oder `s[int i];`.
`KoordList(` `QListWidget *par = 0);`	Falls die Liste in einem QListWidget angezeigt werden soll, muss dieser hier angegeben werden.

`void addItem(KoordItem k);`
`void addItem(float ix,`
`float iy, float iz);`
	Fügt ein Item hinzu, welches entweder schon existiert oder neu erstellt wird.

`bool remove(KoordItem k);`	Entfernt ein Item k, falls eines mit den gleichen Koordinaten existiert.
`bool remove(int i);`	Entfernt das Item an Position i.
`int count();`	Anzahl der Items.
`void clear();`	Leert die Liste.
`int indexOf(KoordItem k);`	Gibt die Postition eines Items mit den Koordinaten von k zurück.

A.1.3 Node

Eine Node ist ein Punkt der graphischen Darstellung. Entweder kann er selbst angezeigt werden oder als Anker für eine Edge dienen oder beides.

`float x;`
`float y;`
`float z;`
`float b;`
	Die räumliche Position x, y und z. b wird für die Homogenisierung des Punktes verwendet.

`int type;`	ColorType, wird durch das KoordItem zugewiesen.
`float p;`	Parameter p, kann frei verwendet werden für die grafische Darstellung.
`QString name;`	Name wird für die 3D-Anzeige verwendet und als Identifikation zum Erzeugen von Edges.

`Node();`
`Node(float x, float y,`
`float z, QString name = ");`
	Erstellt leere oder komplett initialisierte Nodes.

`QPoint flat();`	Gibt die (x, y)-Position zum Zeichnen zurück.

A.1.4 Edge

Edges verbinden zwei Nodes und so lässt sich eine Linie zeichnen. Im allgemeinen werden ihre Eigenschaften anhand ihrer Nodes zugewiesen.

`Edge(Node *a, Node *b,` `int type = 0);`	Hier kann der ColorType manuell festgelegt werden.
`QLine toLine();`	Gibt eine Linie zurück, welche gezeichnet werden kann.

A.1.5 NodeList

`Node matrixMultiply(Node t, float m[]);` `NodeList doTransform(float m[]);`	Wird intern durch die mathematischen Funktionen verwendet.
`QList<Node> nodes;` `QList<Edge> edges;`	Liste der Nodes und Edges.
`void addNode(Node node);` `void addNode(float x, float y, float z, QString name, int type = 0, float p = 0);`	Eine Node kann fertig existieren und hinzugefügt werden oder neu erzeugt werden. ColorType sollte festgelegt werden. p kann verwendet werden für die grafische Darstellung, z.B. Farbverläufe.
`bool addEdge(QString ta, QString tb, int type = 0);`	Die Strings müssen die gleichen sein, wie die von Nodes, die schon in der Liste existieren. Ansonsten Probleme wegen der Pointer-Eigenschaft der Nodes.
`void clear();`	Löscht die Liste.

`NodeList translate(float dx, float dy, float dz);`
`NodeList scale(float fx, float fy, float fz);`
`NodeList rotateX(float beta);`
`NodeList rotateY(float beta);`
`NodeList rotateZ(float beta);`
	Diese Transformatoren können jederzeit verwendet werden.

`NodeList projectParallel( float delta);` `NodeList projectCentral( float front);`	Diese Transformatoren sollten als Letztes vor dem Zeichnen angewendet werden und zwar in einer eigens hierfür kopierten NodeList.
`void copyFrom(NodeList source);`	Diese Funktion übernimmt den Inhalt einer anderen NodeList.