Inhalt

1 Einleitung

Da sich diese Arbeit mit Elektroakustik beschäftigt und speziell das Richtungshören des Menschens unter Simulationsbedingungen untersucht, finde ich es angebracht, den ersten Abschnitt der Diplomarbeit den Grundlagen der Akustik zu widmen.

Ich werde deshalb das zweite Kapitel dazu nutzen, die wichtigsten physikalischen Größen, mit denen der Schall beschrieben wird, zu beschreiben.

Da diese Größen allesamt objektive und messbare physikalische Größen sind, das menschliche Hören aber etwas Subjektives ist, ist es wichtig, auch auf die subjektiven Größen einzugehen (Lautstärkeempfindung, Tonhöheempfindung etc.), vor allem da diese oft nicht proportional zu den messbaren physikalischen Größen sind.

Nach Erläuterung der wichtigsten Elektroakkustischen Größen werde ich im dritten Kapitel auf das Richtungshören des Menschens eingehen, da sich meine Arbeit im Wesentlichen damit auseinander setzt und dabei die Begriffe *Lokalisation* und *HRTF* (***H**ead **R**elated **T**ransfer **F**unctions*) ausführlich beschreiben.

Schließlich werde ich im vierten Kapitel gründlich auf meine eigene Aufgabenstellung eingehen.

Hintergrund dieser Arbeit ist das GUIB-Projekt, das zur Ziel hat, blinden Personen die Arbeit mit dem Computer zu erleichtern. GUIB steht für ***G**raphical **U**ser **I**nterface for **B**lind **P**ersons.*

In diesem Projekt wird versucht, blinden Menschen die Orientierung auf dem Bildschirm mit Hilfe von so genannten *Earcons* (auditive Repräsentation von Bildschirm-Icons) zu ermöglichen, die diese aus verschiedenen Richtungen wahrnehmen.

Je nachdem, aus welcher Richtung ein Geräusch kommt, kann aus dieser Richtung ein Ereignis auf dem Bildschirm zugeordnet werden (z.B. das Öffnen oder schließen eines Fensters, Drücken der Rechten oder Linken Maustaste u.v.m.)

Dabei wird eine Karte benutzt (die Beachtron Karte), die die Geräusche so umrechnet und mittels Kopfhörer so ausstrahlt, das sie aus den verschiedenen Richtungen zu kommen scheinen. Diese Karte wird mit der Programmiersprache C angesteuert und läuft nur auf älteren Rechnern problemlos. Deshalb habe ich die ganze Zeit auf einem 486er mit Win 95 gearbeitet und die beiden selbst geschriebenen Programme unter MS-DOS getestet und verwendet. Ausführliche Informationen zu der Beachtron Karte folgen später.

2 Grundlagen der Akustik

2.1 Schall

Physikalisch gesehen ist Schall eine Welle. In flüssigem und gasförmigem Medium ist Schall immer eine Longitudinalwelle. In Festkörpern können auch Transversalwellen vorkommen.

Schallwellen transportieren wie jede andere Welle Energie und können in Abhängigkeit vom Signal auch Information transportieren. In der Luft beträgt die Schallgeschwindigkeit bei einer Temperatur von 20 °C 343 m/s. Die beiden Energieformen, zwischen denen beim Schall die Energie hin und her pendelt, sind Kompressionsenergie und Bewegungsenergie.

Die wichtigsten Schallfeldgrößen sind der **Schalldruck p(t)** in N/m² und die **Schallschnelle v(t)** in m/s.

Der Schalldruck ist eine ungerichtete Größe, d.h. ein Skalar. Man unterscheidet den Augenblickswert p(t) und den Effektivwert peff.

Die Schallschnelle v(t) dagegen ist eine vektorielle Größe. Sie darf nicht mit der Schallgeschwindigkeit verwechselt werden.
Die Schallschnelle gibt nämlich an, mit welcher Wechselgeschwindigkeit die Luftteilchen um ihre Ruhelage hin und her schwingen. Diese Wechselgeschwindigkeit ist sehr viel kleiner als die Schallgeschwindigkeit und beträgt z.B. bei einem „Lärm“ von 120 dB (Schmerzschwelle) gerade einmal 0.05 m/s (Schallschnelleamplitude). Bei der Hörschwelle des Menschens hat die Schallschnelleamplitude sogar nur noch einen Wert von $5 \cdot 10^{-8}$ m/s.

Man unterscheidet bei der Schallschnelle ebenfalls zwischen Augenblickswert v(t) und Effektivwert veff.

Der kleinste Schalldruck, den das menschliche Ohr eben noch wahrnehmen kann beträgt peff= $10^{-4} \mu b$. Der größte Schalldruck, der ohne Schmerzen ertragen werden kann, liegt bei peff=10^3 μb.

Damit dieser große Zahlenbereich in einem einzigen Diagramm überhaupt dargestellt werden kann, verwendet man für den Schalldruck und für die Schallschnelle eine logarithmische Skala. Logarithmen können aber nur von dimensionslosen Größen gebildet werden, daher wird der effektive Schalldruck durch dividieren durch den Bezugsschalldruck in eine reine Zahl umgewandelt. Das gleiche macht man auch bei der Schallschnelle, indem man sie durch eine Bezugsschallschnelle teilt.

Definition:

$$L_p = 20 \cdot \log_{10}(\frac{p_1}{p_0})dB = 10 \cdot \log_{10}(\frac{p_1^2}{p_0^2})dB \quad (2.1)$$

$$p_0 = 2 \cdot 10^{-5}\, Pa \quad (2.2)$$

$$L(v) = 20 \cdot \log_{10}(\frac{v_1}{v_0})dB \quad (2.3)$$

$$v_0 = 5{,}0 \cdot 10^{-8}\, \frac{m}{s} \quad (2.4)$$

In der Akustik spielt der Schalldruck, bzw. der Schalldruckpegel gegenüber der Schallschnelle eine viel größere Rolle, da er eine sehr leicht messbare skalare Größe darstellt. Die Schallschnelle dagegen ist wegen ihrem vektoriellen Charakter viel schwerer zu messen.

Man teilt den Schall je nach Frequenz in folgende Bänder ein:

- Infraschall < 16 Hz ist für Menschen nicht hörbar, viele Tiere verständigen sich in diesem Band (z.B. Wale), wird auch im Kino eingesetzt, um ein Angstgefühl im Publikum zu erzeugen, da man mit Infraschall die Bauchgegend des Körpers zum Vibrieren bringen kann.
- Hörschall von 16 Hz bis 20 kHz, ist für Menschen hörbarer Schall
- Ultraschall von 20 kHz bis 10 GHz ist für Menschen nicht hörbar, wird ebenfalls von vielen Tieren genutzt (z.B. Fledermäusen)
- Hyperschall > 10 GHz sind nur noch bedingt ausbreitungsfähige Wellen

2.2 Zusammenhang von Schallenergiedichte, Schallleistung und Schallintensität

In einer ebenen Schallwelle, deren Schalldruck den Effektivwert peff hat, fließt durch einer Fläche der Größe A, welche Senkrecht zur Fortpflanzungsrichtung der Welle steht, die Schallleistung:

$$P = 2{,}4 \cdot 10^{-9}(\frac{A}{cm^2}) \cdot (\frac{p_{eff}}{\mu b})^2 Watt \quad (2.5)$$

Hieraus wird die ***Schallintensität*** abgeleitet:

$$J = \frac{P}{A} = 2{,}4 \cdot 10^{-9} \cdot (\frac{p_{eff}}{\mu b})^2 Watt \quad (2.6)$$

Man kann die ***Schallintensität*** auch anders ausdrücken:

$$J = \frac{1}{T} \cdot \int_0^T p(t) \cdot v(t) dt \qquad (2.7)$$

Im Schallfeld einer ebenen Welle ergibt sich die ***Schallintensität*** aus dem Produkt der Effektivwerte von Schalldruck und Schallschnelle:

$$J = p \cdot v = \frac{p^2}{Z} = Z \cdot v^2 = \zeta^2 \cdot \omega^2 \cdot Z = \frac{a^2 \cdot Z}{\omega^2} = E \cdot c = \frac{P_{ak}}{A} \qquad (2.8)$$

Schalldruck p in N/m² = Pa
Schallschnelle v in m/s
Schallkennimpedanz Z in N·s/m³
Schallauslenkung der Luftteilchen ξ *in m*
Kreisfrequenz $\omega = 2\,\pi \cdot f$ in Hz
Schallbeschleunigung a in m/s²
Schallenergiedichte E in W·s/m³
Schallgeschwindigkeit c in m/s
Durchströmte Fläche A in m², P_{ak} die akustische Leistung
Bei einer ebenen fortschreitenden Welle gelten für die Schallleistung folgende Zusammenhänge:

$$P_{ak} = \frac{p^2 \cdot A}{Z} = Z \cdot v^2 \cdot A = \zeta^2 \cdot \omega^2 \cdot Z \cdot A = \frac{a^2 \cdot Z \cdot A}{\omega^2} = E \cdot c \cdot A \qquad (2.9)$$

Hierbei ist:

- Auslenkung der Luftteilchen ξ in m.

- Kreisfrequenz ω
- Frequenz f in Hz
- Akustische Impedanz = Schallkennimpedanz ($Z = c \cdot \rho$)
- Dichte der Luft (des Mediums) = Luftdichte ρ in kg/m³
- Schallgeschwindigkeit c in m/s
- Schallkennimpedanz Z in N·s/m³
- Schallschnelle v in m/s
- Schallintensität J in W/m²
- Schallbeschleunigung a in m/s²
- Schallenergiedichte E in W·s/m³
- Durchströmte Fläche A in m²

Man kann die logarithmische Gleichung für den Schallpegel auch in Intensitäten ausdrücken:

$$L(J) = 10 \cdot \log_{10}(\frac{J_1}{J_0})dB = 20 \cdot \log_{10}(\frac{p_1}{p_0})dB \qquad (2.10)$$

$$J_0 = 10^{-12} \frac{W}{m^2} \qquad (2.11)$$

Der Schalleistungspegel ist folgendermaßen definiert:

$$L_\omega = 10 \cdot \log_{10}(\frac{\omega_1}{\omega_2})dB \qquad (2.12)$$

$$\omega_0 = 10^{-12} W \qquad (2.13)$$

2.3 Subjektive Schallgrößen:

2.3.1 Lautstärke:

Die Lautstärke ist ein Maß für das subjektive Empfinden einer objektiv messbaren Größe, nämlich des Schalldruckpegels, oder der Schallintensität.

Bei der Lautstärke handelt es sich um eine psychoakustische Größe, deren Einheit in Phon angegeben wird.

Das menschliche Ohr nimmt Töne derselben Schallintensität J bei unterschiedlichen Frequenzen unterschiedlich Laut wahr.

Um diese wichtige psychoakustische Eigenschaft des Gehörs zu berücksichtigen, hat man folgende Vereinbarung getroffen:

Bei einer Schall-Frequenz von 1000 Hz sollen Schalldruckpegel, gemessen in dB und Lautstärkepegel, gemessen in Phon, zahlenmäßig übereinstimmen.

Die Messmethode für die Lautstärke ist einfach: Der Schalldruck eines Tones wird so eingestellt, das er als gleich laut wie ein 1 kHz Referenzton wahrgenommen wird.

Die Abbildung 2.1 verdeutlicht den Zusammenhang zwischen Schalldruckpegel und Lautstärkepegel.

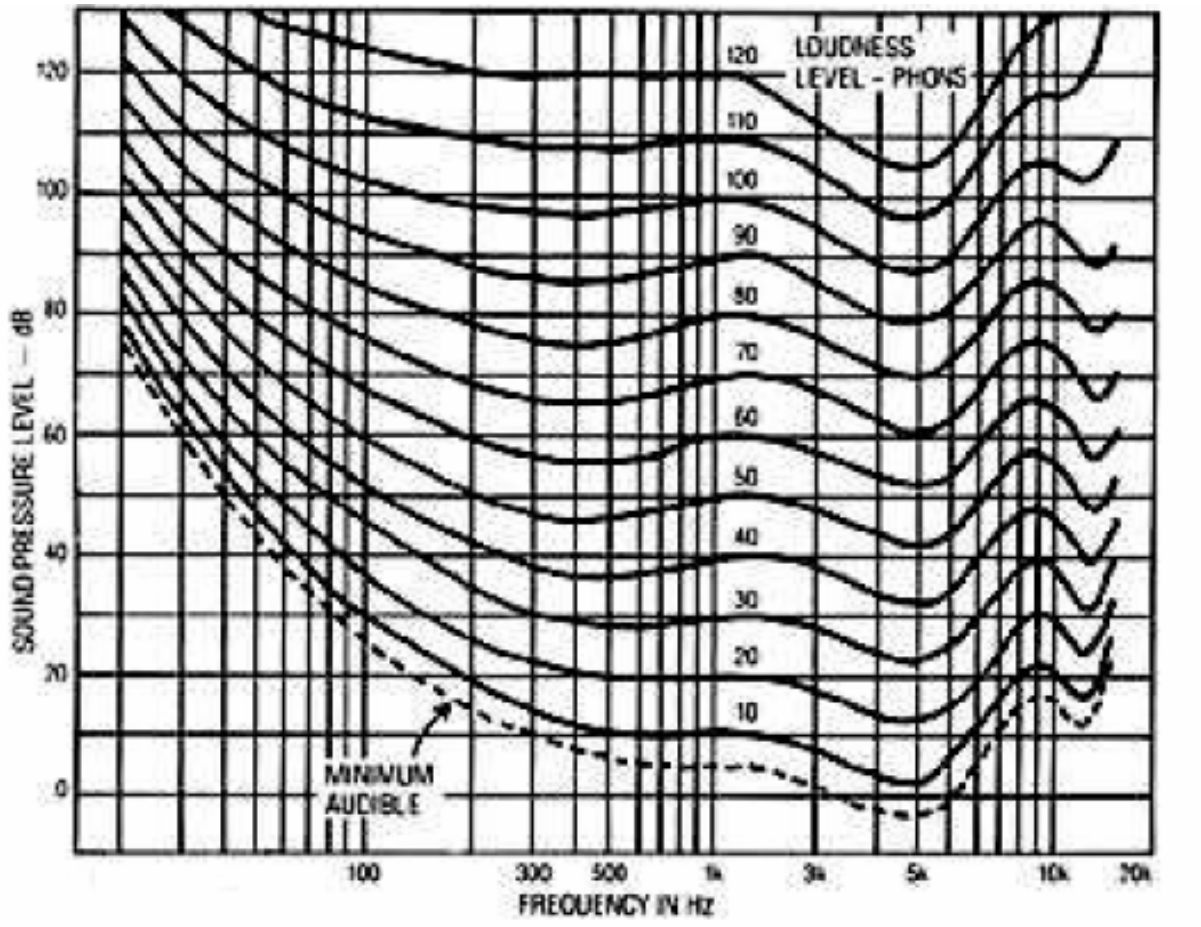


Abbildung 2.1: Kurven gleicher Lautstärke

Man kann z.B. an der obigen Abbildung erkennen, dass ein 100 Hz Ton mit einem Schalldruckpegel von 50 dB vom menschlichen Gehör genauso laut wahrgenommen wird, wie ein 1000 Hz Ton mit einem Schalldruckpegel von 40 dB. Bei beiden Tönen ist die Lautstärke 40 Phon.

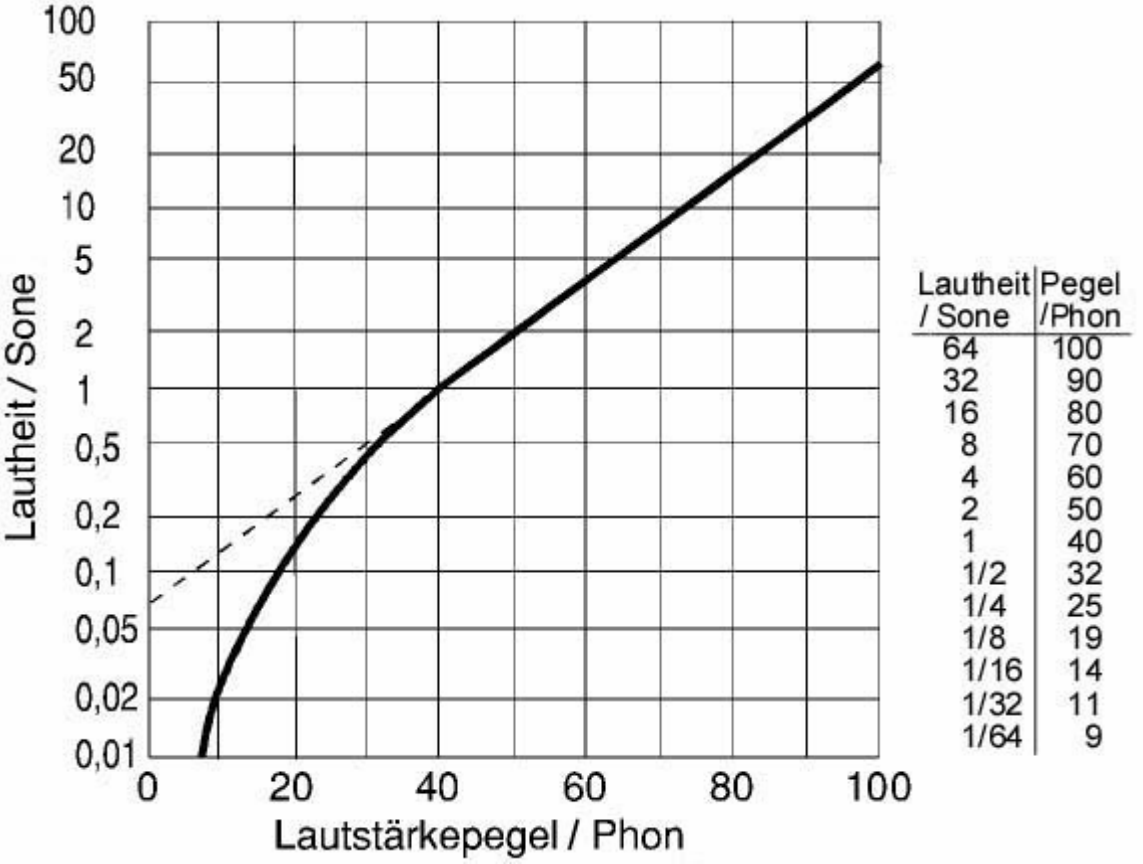
2.3.2 Lautheit:

Die Lautheit gibt ebenfalls an, wie laut Schall subjektiv empfunden wird und wird in der Einheit Sone angegeben. Die Lautheit verdoppelt sich, wenn der Schall als doppelt so Laut empfunden wird. Sie ermöglicht somit dem Anwender das Vergleichen zweier Schallereignisse.

Definition:

Ein Sone ist definiert als die empfundene Lautstärke eines Schallereignisses von 40 phon. Bei mittleren und hohen Lautstärken führt eine Erhöhung der Lautstärke von 10 phon zu einer Verdopplung der Lautheit.

Abbildung 2.2 soll den Zusammenhang zwischen Sone und Phon verdeutlichen:



Lautheit / Sone	Pegel /Phon
64	100
32	90
16	80
8	70
4	60
2	50
1	40
1/2	32
1/4	25
1/8	19
1/16	14
1/32	11
1/64	9

Abbildung 2.2: Zusammenhang zwischen Sone und Phone

2.3.3 Tonheit

Die Tonheit ist eine weitere subjektive Größe, mit der man die Tonhöheempfindung des Menschen angibt. Die Einheit der Tonheit ist *mel*.

Die Tonheitsskala wurde durch psychoakustische Versuche ermittelt, bei denen man Versuchspersonen gefragt hat, welche Frequenzabstände eine Verdoppelung oder Halbierung der Tonhöheempfindung zur folge haben. Hat man durch diese Methode genügend Werte ermittelte, kann man durch Interpolation die Tonheit als stetige Funktion der Frequenz angeben.

Dem Ton mit der Frequenz von f= 131 Hz schreibt man willkürlich eine Tonheit von 131 mel zu. Eine Verdoppelung der Tonhöheempfindung hat auch eine Verdoppelung der Tonheit zur Folge.

Abbildung 2.3 zeigt den Zusammenhang zwischen der Tonheit und der Frequenz.

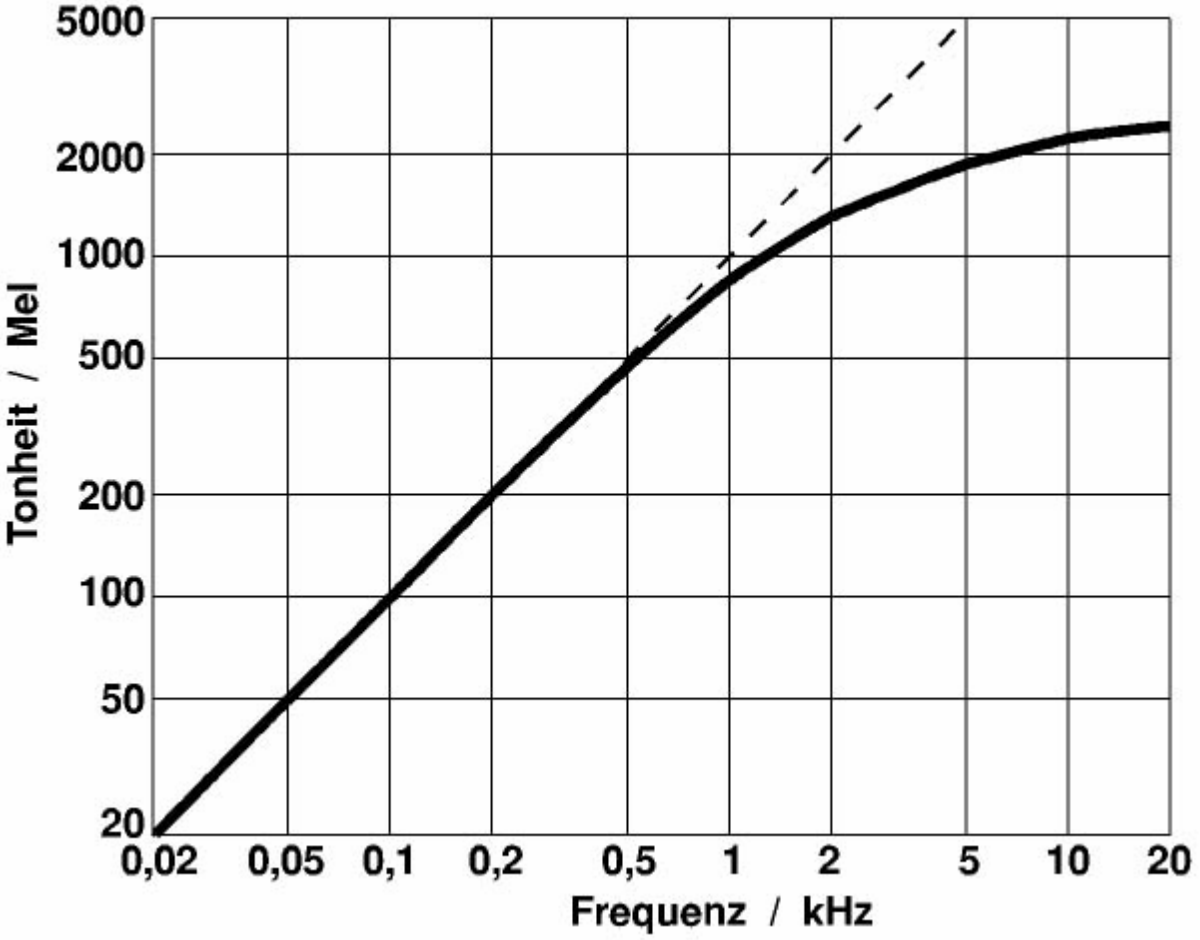


Abbildung 2.3: Zusammenhang zwischen Tonheit und Frequenz

Man sieht, dass unterhalb einer Frequenz von 500 Hertz ein proportionaler Zusammenhang zwischen der Tonheit und der Frequenz besteht. Erst oberhalb davon gibt es eine deutliche Abweichung.

2.4 Die Hörfläche

Die Hörfläche ist ein Gebiet, indem das menschliche Ohr Schall wahrnehmen kann. Sie wird von vier Grenzen eingeschlossen, nämlich der höchsten und niedrigsten noch hörbaren Frequenz und der niedrigsten und höchsten Schwelle des Schalldruckpegels, indem das menschliche Ohr noch ohne Schmerzen den Schall wahrnehmen kann.

Die Hörfläche ist altersabhängig und schrumpft mit steigendem Alter. Besonders die hohen Frequenzen und die niedrigen Schalldruckpegel werden mit steigendem Alter immer schlechter wahrgenommen.

Der Begriff Schall lässt sich auch mit der Hörfläche erklären, denn Schall ist all das, was in die Hörfläche hineinfällt und somit für uns hörbar ist.

Die Abbildung 2.4 zeigt die Hörfläche:

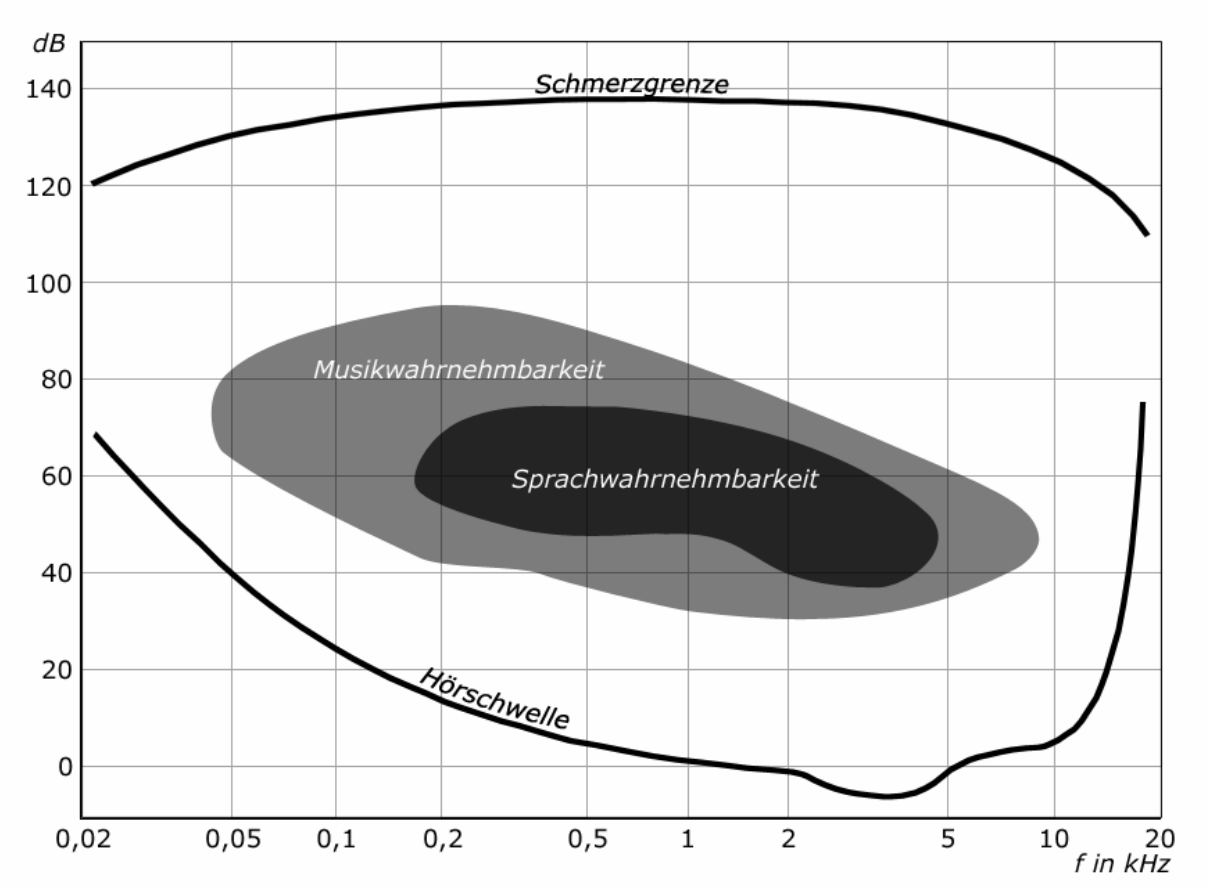


Abbildung 2.4: Hörfläche

Wie man sehen kann, gibt es drei Zonen in der Hörfläche: die größte Zone für die allgemeine Hörbarkeit von Schall, indem alles hineinfällt, was unser Ohr wahrnehmen kann, eine etwas kleinere Zone für die Wahrnehmbarkeit von Musik und eine noch kleiner Zone für die Wahrnehmbarkeit von Sprache.

2.5 Verdeckung (Maskierung)

Wie man an der Hörfläche sehen kann, gibt es eine untere Hörschwelle, die angibt, welche Schalldruckpegel in Abhängigkeit der Frequenz vom Menschen gerade noch wahrgenommen werden können. Diese Hörschwelle, auch *absolute Hörschwelle* genannt, wurde in einer Umgebung gemessen, in der es sonst absolut still war und keine Störgeräusche vorhanden waren.

Es wäre nun auch interessant, herauszufinden, wie sich die Hörschwelle ändert, wenn neben dem Testton, dessen Hörschwelle ermittelt werden soll, auch noch ein Störgeräusch vorhanden ist. Die Abbildung 2.5 zeigt eine solche Messung, bei der als Störschall ein 1 kHz Sinuston verwendet wurde.

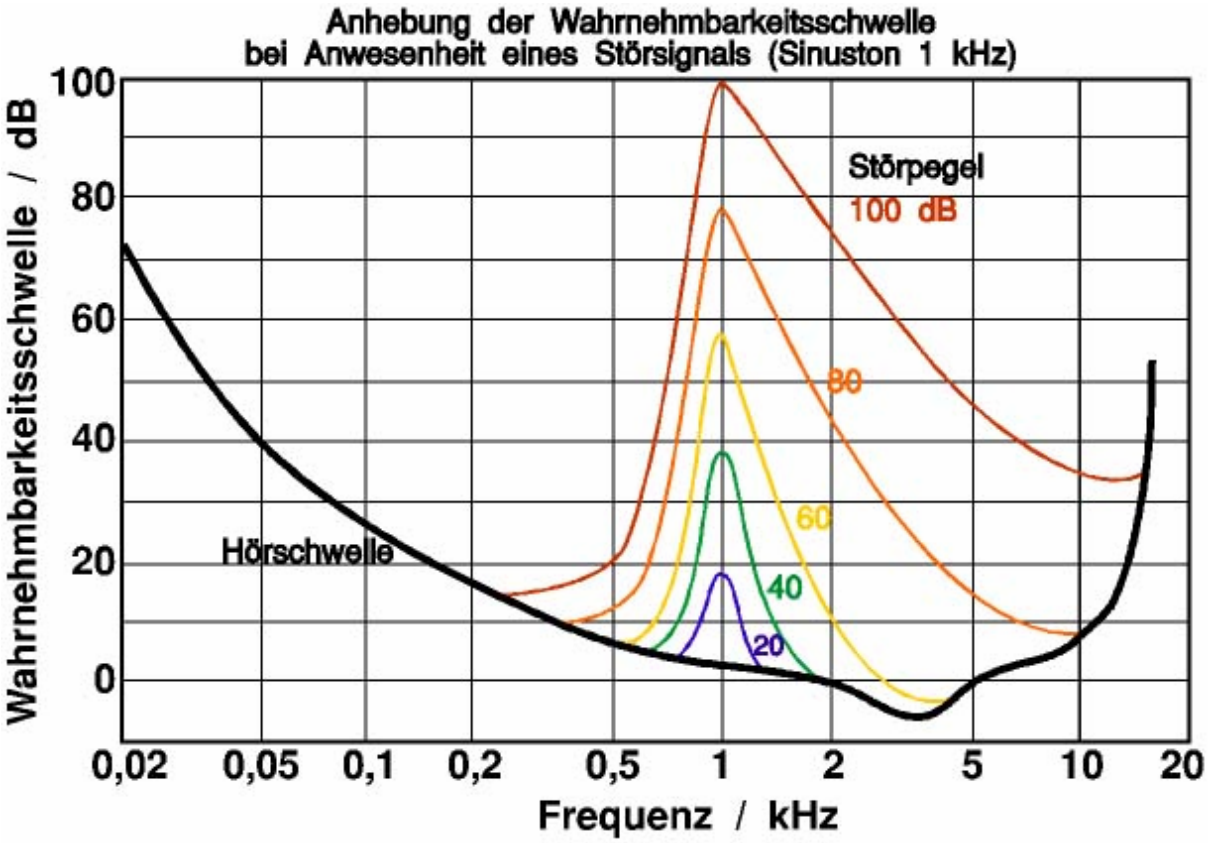


Abbildung 2.5: Wirkungsweise von Maskierungseffekten

Wie man in der obigen Abbildung sieht, liegt die Mithörschwelle in der Nähe der Störfrequenz weit oberhalb der Ruhehörschwelle.

Als Störschall kommen natürlich auch andere Geräusche wie Rauschen (Weißes Rauschen, Hochpass-, Tiefpass-, Bandpassrauschen) oder Klänge in Frage. Dabei gibt es zu jedem dieser Geräusche eine besondere Mithörschwelle. Im Allgemeinen kann man aber sagen, dass die Mithörschwelle in dem Frequenzgebiet, den der Störschall einnimmt, weit oberhalb der Ruhehörschwelle verläuft.

Kenntnisse über die Maskierungseigenschaften des menschlichen Gehörs sind interessant um zum Beispiel bei der MP3 Codierung solche Frequenzanteile für den Moment auszufiltern, indem sie wegen der Maskierung unhörbar sind, bzw. teilweise maskierte Frequenzanteile mit einer geringeren Datenrate zu übertragen. Eine ausführliche Beschreibung der Maskierungseigenschaften des menschlichen Gehörs würde den Rahmen dieser Arbeit sprengen. Deshalb belasse ich meine Schilderungen hierbei.

2.6 Frequenzgruppen

Mit Frequenzgruppen bezeichnet man die Eigenschaft des Gehörs, den Schall in Frequenzbereiche aufzuteilen, die gemeinsam ausgewertet werden. Das Gehör teilt die Hörbaren Frequenzen in ca. 24 Frequenzgruppen ein.

Zur genauren Erläuterung der Entstehung von Frequenzgruppen ziehe ich Abbildung 2.6 zur Hilfe:

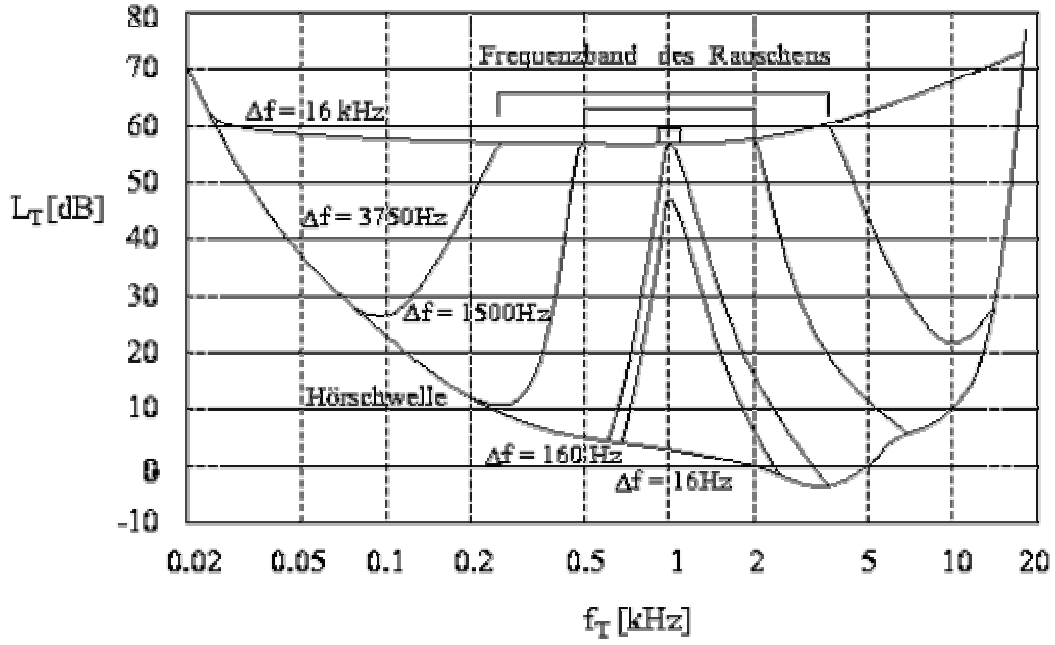


Abbildung 2.6

Die Abbildung 2.6 zeigt die Mithörschwellen, die entstehen, wenn Bandpassrauschen variabler Bandbreite und konstanter Mittenfrequenz als maskierendes Störsignal benutzt wird. Die Mittenfrequenz des Bandpassrauschens liegt dabei immer bei 1 kHz. Die Bandbreite variiert von Δf= 16 Hz bis Δf=16 kHz. Der Schallintensitätsdichtepegel des Bandpassrauschens wurde dabei immer konstant auf L_R=40 dB gehalten.

Man sieht nun, dass innerhalb des Frequenzbereiches des Bandpassrauschens die Mithörschwellen konstant bleiben und alle denselben Wert haben. Erst außerhalb dieses Bereiches fangen sie an, recht flach abzufallen. Der flache Abfall liegt nicht dadurch begründet, dass ein schlechter Filter zu Erzeugung des Bandpassrauschens benutzt wurde, denn das Leistungsdichtespektrum des Bandpassrauschens zeigte eine fast ideale Rechteckform, sondern ist eine Eigenschaft des Gehörs. Die Knickpunkte der Mithörschwellen stimmen allerdings recht gut mit der oberen und unteren Grenzfrequenz des Bandpassrauschens überein.

Wenn nun die Bandbreite des Bandpassrauschens verringert wird, sieht man, dass die zunächst trapezförmige Gestalt der Mithörschwellen kleiner wird, d.h. dass die Flanken auf beiden Seiten näher beieinander rücken, bis bei einer kritischen Bandbreite Δf_{FG} die trapezförmige Gestalt in eine dreieckige Gestalt übergeht. Es muss noch betont werden, dass eine Verringerung der Bandbreite mit einer Verringerung der Leistung des Rauschens einhergeht, da der Schallintensitätsdichtepegel konstant gehalten wird.

Die Spitze des Dreiecks liegt dabei immer bei der Mittenfrequenz $f_T = f_M$. Wird nun die Bandbreite weiter verringert, dann sinkt von nun an die Mithörschwelle für die Frequenz $f_T=f_M$ **unter** das bisherige Niveau der Mithörschwelle.

Für andere Mittenfrequenzen des Bandpassrauschens ergeben sich analoge Ergebnisse, jedoch ändert sich mit der Mittenfrequenz auch die kritische Bandbreite Δf_{FG} und ist somit Frequenzabhängig ($\Delta f_{FG}(f)$).

Dieses Verhalten des menschlichen Gehörs zeigt, dass im Ohr Schallintensitäten zu kritischen Bandbreiten, den so genannten Frequenzgruppen zusammengefasst werden und diese zur Bildung der Mithörschwellen herangezogen werden.

Zu jeder Frequenz innerhalb der Hörfläche gibt es eine zugehörige Frequenzgruppe $\Delta f_{FG}(f)$. Dabei ist der Zusammenhang zwischen der Frequenzgruppe Δf_{FG} und der Frequenz f nichtlinear aber monoton.

3 Psychoakustik des räumlichen Hörens

3.1 Schallereignis vs. Hörereignis

Das Wort „Schall“ wird ausschließlich für die physikalische Seite des Hörvorgangs verwendet und zwar oft in der Zusammensetzung „Schallereignis“. Dagegen wird das Wort „Hörereignis“ für das vom menschlichen Ohr Gehörte, bzw. das Wahrgenommene verwendet.

Die Trennung von *Schall- und Hörereignissen* ist notwendig, um den Unterschied zwischen physikalischen Phänomenen und Phänomenen der Wahrnehmung zum Ausdruck zu bringen.

Es gibt eine assoziative Beziehung zwischen Schall- und Hörereignissen. Die Abbildung 3.1 verdeutlicht diese Beziehung:

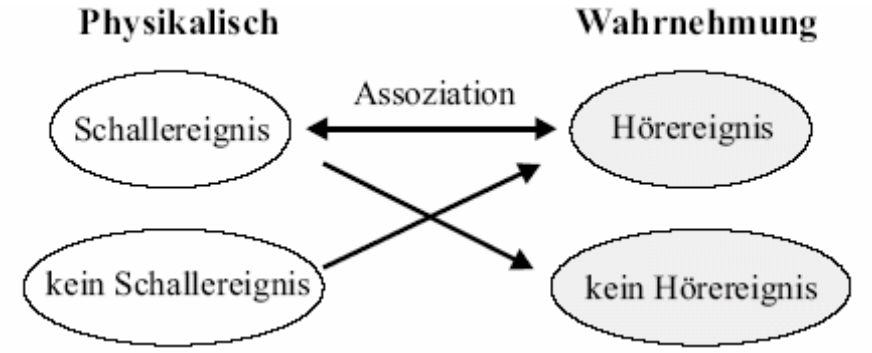


Abbildung 3.1: Assoziation zwischen Schall- und Hörereignissen ([Art-diss])

Wie man sieht, kann ein Schallereignis zu einem dazugehörigen Hörereignis führen. Aber nicht jedes Schallereignis wird unbedingt wahrgenommen (z.B. bei Schwerhörigkeit). Man sieht aber auch, dass ein Hörereignis auftreten kann, ohne das dem ein Schallereignis vorausgeht (z.B. bei Hörsturz oder Tinitus).

3.2 Lokalisation und Lokalisationsunschärfe

Ein wichtiger Parameter zur Beurteilung des räumlichen Hörens gewinnt man, indem man den Ort der Schallquelle mit dem wahrgenommenen Hörereignisort vergleicht.

Um diesen Vergleich zu erleichtern, wurden die folgenden zwei Definitionen eingeführt [Bla83]:

Lokalisation: Zuordnungsgesetz oder –regel (Operator) zwischen dem Ort eines Hörereignisses (z.B. Richtung und oder Entfernung) und bestimmten Merkmalen eines Schallereignisses (z.B. Zuordnung von Hörereignis- und Schallquellenort).

Lokalisationsunschärfe: Kleinste Änderung eines bestimmten Merkmals oder bestimmter Merkmale des Schallereignisses, die gerade zu einer Ortsänderung des Hörereignisses (z.B. Richtung und/oder Entfernung) führen. Diese wird auch als MAA(„minimum audible angle“) bezeichnet [MG91].

Da die Lokalisationsunschärfe empirisch ermittelt werden muss, wird in der Psychoakustik folgende Regel angewendet: Lokalisationsunschärfe ist die Ortsänderung der Schallquelle, bei der gerade 50% der Versuchspersonen eine Änderung des Hörereignisortes bemerken [Bla83].

Diese beiden Definitionen führen zu zwei Teilfragen:

1. Wo erscheint das Hörereignis bei gegebenem Schallquellenort? (Lokalisation)
2. Wie groß muss die Ortsänderung der Schallquelle mindestens sein, damit das Hörereignis seinen Ort gerade ändert? (Lokalisationsunschärfe)

3.3 Kopfbezogenes Koordinatensystem

Um Ortsangaben zum räumlichen Hören machen zu können (z.B. Schallquellenort, Hörereignisort, Messpunkt usw.) wird in der Regel ein *kopfbezogenes* Koordinatensystem von Kugelkoordinaten verwendet. Kopfbezogen heißt, dass das Koordinatensystem sich bei Bewegungen des Kopfes der Versuchsperson mitbewegt. Da Menschen ihre Ohren nicht relativ zum Kopf bewegen können, ist es auch gleichzeitig ohrenbezogen. Die Abbildung 3.2 zeigt ein kopfbezogenes Koordinatensystem:

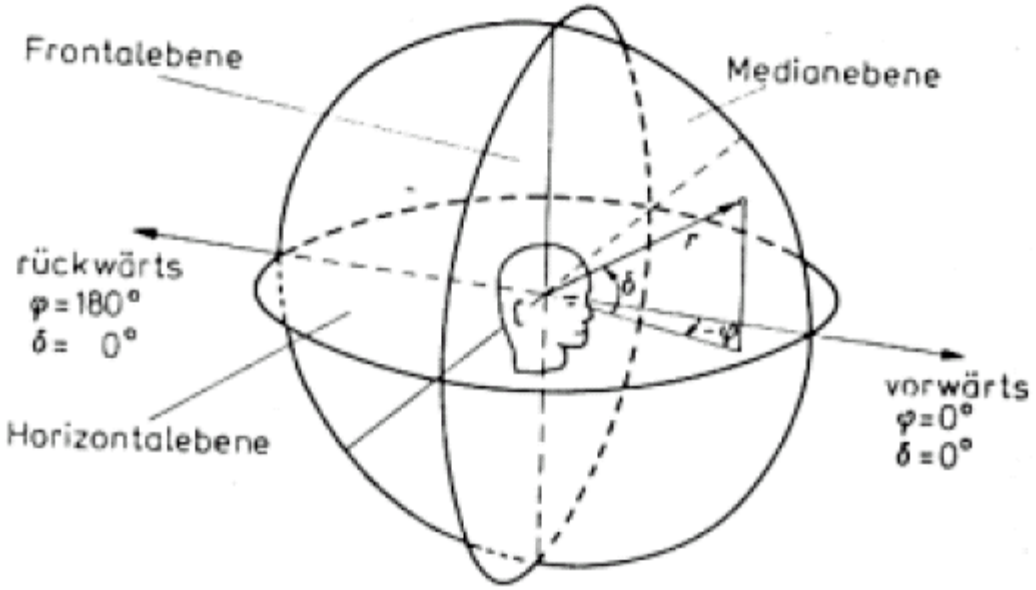


Abbildung 3.2: Kopfbezogenes Koordinatensystem; r: Entfernung, φ: Seitenwinkel (Azimuth), δ: Erhebungswinkel (Elevation) ([Bla83])

3.4 Schallfeld

Beliebige Schallfelder lassen sich in Elementarfelder zerlegen. Dabei kann man das Prinzip von Huygens und Fresnel benutzen, nachdem jeder Punkt eines beliebigen Wellenfeldes als Quelle einer Kugelwelle (Elementarwelle) aufgefasst werden kann. Dadurch wird der Wellenvorgang an jedem Punkt innerhalb des Feldes berechenbar. Die Kugelwelle ist eine punktsymmetrische Welle mit folgenden Eigenschaften:

- Wellenparameter sind abhängig von der Entfernung

- Wellenparameter sind unabhängig zu der Ausbreitungsrichtung

Schallquellen, die Kugelstrahlen erzeugen, werden Kugelstrahler 0. Ordnung, Elementarstrahler oder „atmende“ Kugeln genannt. Die erzeugten Schallwellen lassen sich durch die folgenden beiden Gleichungen beschreiben:

$$p(t,r) = R_e\left\{const \cdot \frac{j2\pi f\rho_0}{r} \cdot e^{j2\pi ft} e^{-j2\pi r/\lambda}\right\} \qquad (3.1)$$

$$v(t,r) = R_e\left\{const \cdot (\frac{j2\pi/\lambda}{r} + \frac{1}{r^2}) e^{j2\pi ft} e^{-j2\pi r/\lambda}\right\} \qquad (3.2)$$

p(t,r) bezeichnet den Schalldruckverlauf und v(t,r) bezeichnet den Schallschnelle verlauf. ρ ist die Dichte des Mediums. Wie man an den Gleichungen sieht, fällt die Schalldruckamplitude mit dem Faktor 1/r. Die Amplitude der Schallschnelle fällt im Fernfeld ($r>>2\pi r/\lambda$) auch mit dem Faktor 1/r, während sie im Nahfeld mit dem Faktor $1/r^2$ fällt.

Abbildung 3.3 zeigt die physikalische Anordnung des Schallwellenempfangs. Man sieht, dass sich im Fernfeld näherungsweise eine ebene Welle bildet.

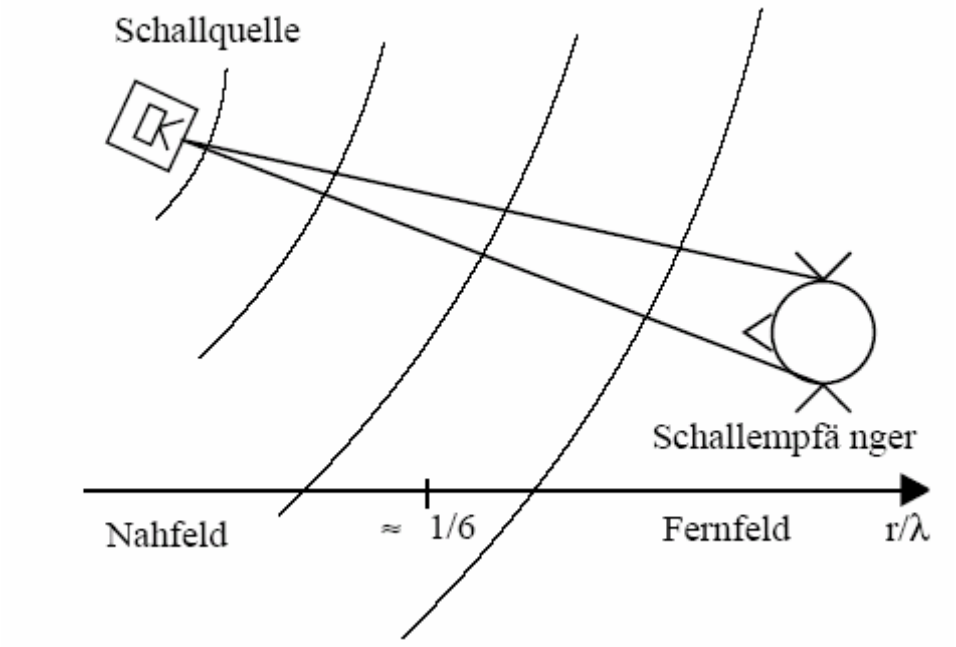


Abbildung 3.3: Physikalische Anordnung des Schallwellenempfangs, Bildung einer näherungsweise ebenen Wellenfront im Fernfeld ([Art-diss])

Die Feldgleichungen des Kugelwellenfeldes gelten mit guter Nährung auch für beliebig andere Strahler (z.B. Lautsprechermembran), wenn folgende Bedingungen erfüllt sind:

1. der Messpunkt muss sich in genügend großem Abstand vom Strahler befinden.
2. die Ausdehnung des Strahlers muss klein gegen die Wellenlänge sein.

3.5 Testsignale

Für Untersuchungen des räumlichen Hörens können beliebig komplizierte Schallfelder eingesetzt werden. Für Untersuchungen eignen sich jedoch besonders Signale mit möglichst elementaren zeitlichen und spektralen Eigenschaften. Daher verwendet man in der Psychoakustik häufig folgende Elementarsignale als Testsignale:

- Stoßklick (schmaler Rechteckimpuls, Impulsbreite<25µs)
- Dauertöne (monofrequente Schwingung)
- Gaußtöne
- Rauschsignale („weiß“, „rosa“, hochpass oder tiefpass gefiltertes weisses Rauschen)

Wie später näher erläutert wird, ist das räumliche Hören *frequenzabhängig*. Abhängig davon, ob die Energie, bzw. Leistung im Zeit- oder Frequenzbereich konzentriert werden soll, eignen sich Stoßklicks oder Dauertöne. Eine breitbandige Erregung des Systems wird durch die Verwendung von „weißem“ Rauschen ermöglicht.

3.6 Hören in der Horizontal- und Medianebene

In diesem Abschnitt möchte ich nun auf die Funktionen einzelner am räumlichen Hören beteiligter Komponenten eingehen. Dabei muss zwischen dem Hören in der Horizontal- und Medianebene unterschieden werden. Zunächst möchte ich das Hören in der Horizontalebene erklären, da es im Vergleich zu Medianebene leichter zu verstehen ist.

3.6.1 Hören in der Horizontalebene

Die Besonderheit des Hörens in der Horizontalebene liegt in der Tatsache, dass die ankommenden Schallsignale sowohl am rechten, als auch am linken Ohr empfangen werden (in der Literatur benutzt man dafür den Begriff *binaurale Empfangskonstellation*).

Je nachdem, von welcher Richtung die Schallwellen kommen, erreichen sie das eine Ohr früher, als das Andere. Dadurch entstehen Laufzeitunterschiede, die physikalisch durch die unterschiedlich langen Wegstrecken zum linken und zum rechten Ohr zu erklären sind.

Diese Laufzeitunterschiede werden in der Fachliteratur mit ITD oder IPD abgekürzt (ITD „interaural time difference“ bzw. IPD „interaural phase difference“).

Die binaurale Empfangskonstellation führt noch zu einem zweiten Effekt, der ebenfalls wichtig für das Hören in der Horizontalebene ist. Der Kopf des Menschen bewirkt nämlich einen

Abschattungseffekt, der zur Folge hat, dass das der Schallquelle abgewandte Ohr eine Amplitudendämpfung erfährt. Diese Intensitätsunterschiede werden in der Fachliteratur mit IID oder ILD abgekürzt (IID „interaural intensity difference“ bzw. ILD „interaural level difference“).Die Abbildung 3.4 soll diese beiden Effekte verdeutlichen:

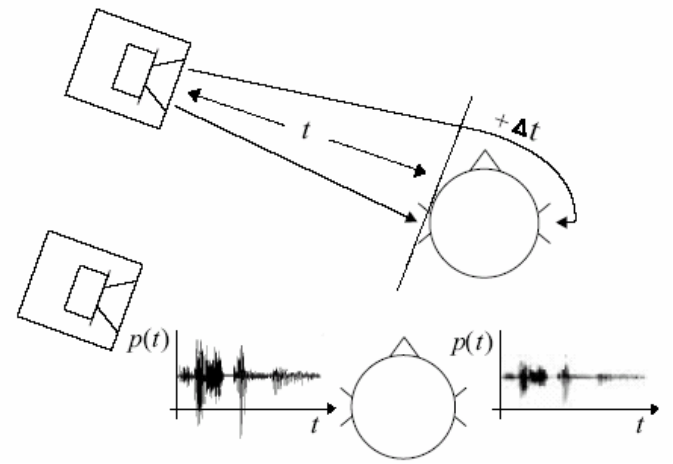


Abbildung 3.4: Interaurale Zeit- und Pegeldifferenzen (ITD/ILD) ([Art-diss])

Wie die Abbildung 3.4 zeigt, kommt das Signal am rechten Ohr später an, als am linken Ohr. Außerdem ist es wegen der Abschattung durch den Kopf auch noch gedämpft. Diese beiden Merkmale können von unserem Gehör ausgewertet werden und sind die wesentlichen Lokalisationsquellen in der Horizontalebene.

Sowohl die interauralen Amplitudendämpfungen, als auch die interauralen Laufzeitunterschiede variieren mit der Richtung des einfallenden Schalls. So kann das der Schallquelle abgewandte Ohr eine Amplitudendämpfung von bis zu 35 dB erfahren [MG91].

Die interauralen Laufzeitunterschiede lassen sich leicht anhand von geometrischen Modellen des Kopfes abschätzen. Die Abbildung 3.5 verdeutlicht eine solche Abschätzung:

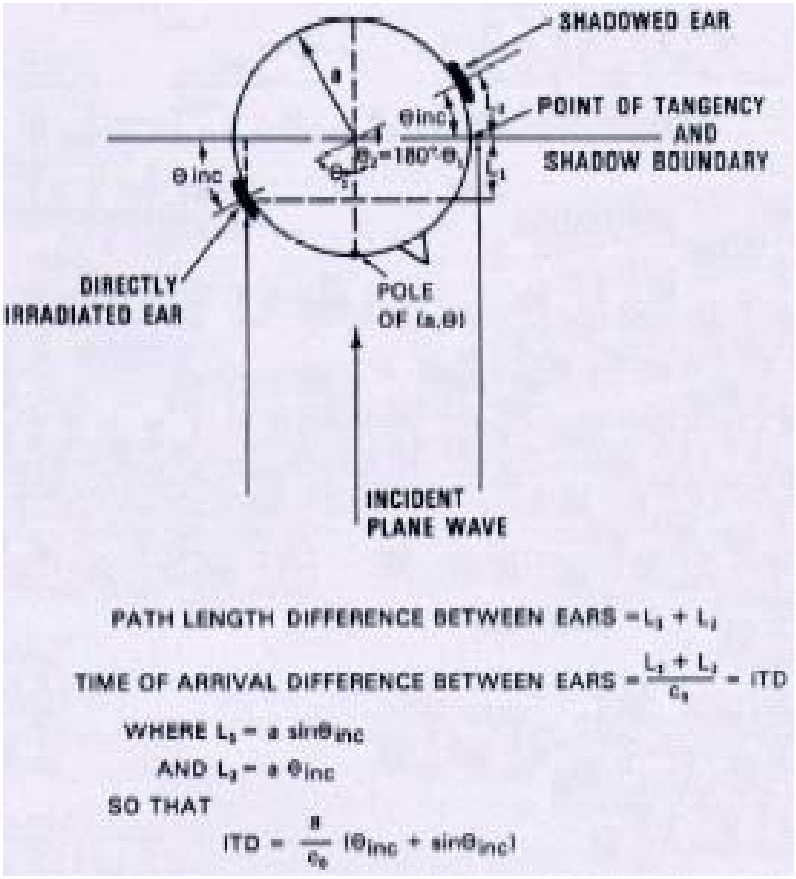


Abbildung 3.5: Abschätzung der Laufzeitunterschiede basierend auf geometrischen Kalkulationen der Kopfabmessungen

Je nach Größe des Kopfes können die interauralen Laufzeitunterschiede bis zu 0.63 ms betragen. Die kleinsten Laufzeitunterschiede, die unser Ohr noch wahrnehmen kann, betragen 0.03 ms, die zu einer Lokalisationsunschärfe von 3°-5° führen.

Die Abbildung 3.6 zeigt den Zusammenhang zwischen Schalleinfallsrichtung und ITD.

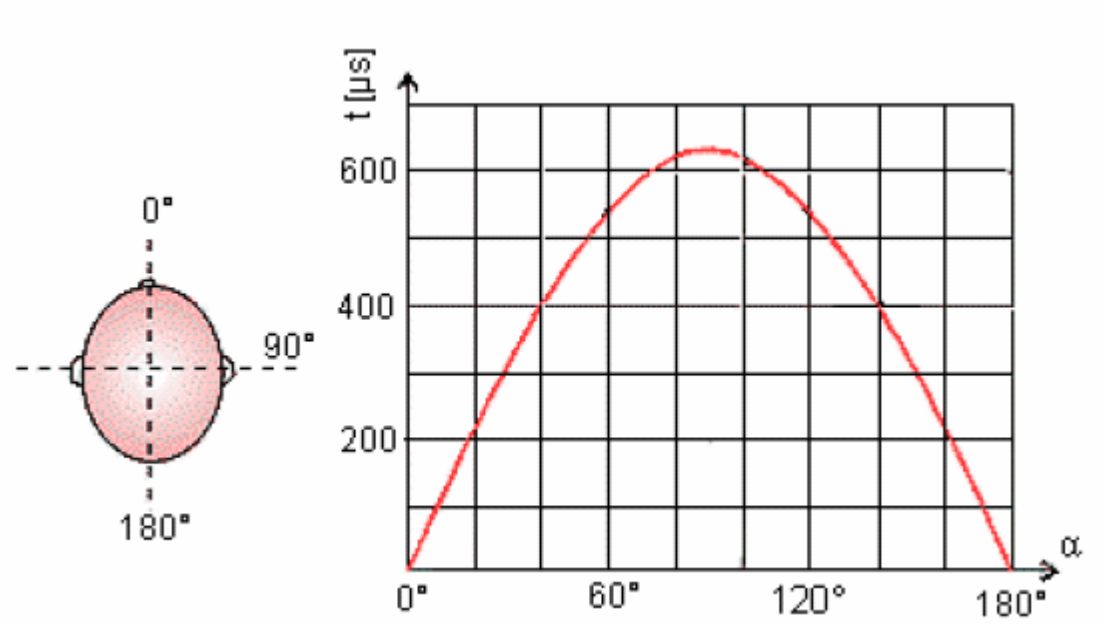


Abbildung 3.6: ITD als eine Funktion des Azimuts ([Wers-diss])

Die interauralen Merkmale sind nicht nur richtungsabhängig, sondern auch frequenzabhängig. Diese Frequenzabhängigkeit kann durch die *interaurale (komplexe) Übertragungsfunktion* H(f) beschrieben werden. Diese ist das Verhältnis der Schalldrücke in den beiden Gehörgängen an einem vergleichbaren Messpunkt.

$$H_I(f) = \frac{P_R(f)}{P_L(f)} = \frac{\hat{p}_r \; e^{j(2\pi f t + \Phi_r(f))}}{\hat{p}_l \; e^{j(2\pi f t + \Phi_l(f))}} = |H_I(f)| \; e^{-j\,b(f)} \tag{3.3}$$

In der obigen Gleichung werden die frequenzabhängigen Phasenverschiebungen am linken und am rechten Ohr durch das Phasenmaß b(f) beschrieben.

3.6.2 Auswertung von ITD und ILD

Die Auswertung von ITD und ILD erfolgt nicht gleichmäßig über das gesamte hörbare Frequenzband, sondern in einem großen Frequenzbereich wird je nach Frequenz eine der beiden Parameter bevorzugt ausgewertet.

Bei tiefen Frequenzen (f < 1000Hz) sind die Abmessungen des Kopfes kleiner als die halbe Wellenlänge des Schalls. Deshalb stellt der Kopf fast kein Hindernis für die ankommenden Wellen dar und das Gehör kann hier die Phasenlaufzeiten von Ohr zu Ohr besonders genau auswerten. Deshalb dominiert die Auswertung von ITD bei niederfrequenten Signalanteilen.

Bei hochfrequenten Signalanteilen dagegen sind die Abmessungen des Kopfes größer als die Wellenlänge des Schalls. Daher kann das Gehör bei diesen Frequenzen die Phasenlaufzeiten nicht mehr exakt bestimmen. Dagegen dominieren in diesem Frequenzbereich die Pegelunterschiede, die dann vom Gehör auch bevorzugt ausgewertet werden.

In einem Frequenzband von 1500-3000 sinkt die räumliche Hörleistung, da in diesem Frequenzbereich die Auswertung der ITD nur noch eingeschränkt möglich ist, und die Wirksamkeit der ILD auch noch mangelhaft ist. Dieser Zusammenhang zwischen ITD und ILD ist in der Fachliteratur auch als „Duplex-Theorie“ bekannt [MG91].

Die Abbildung 3.7 verdeutlicht den relativen Beitrag von ITD und ILD in Abhängigkeit der Frequenz.

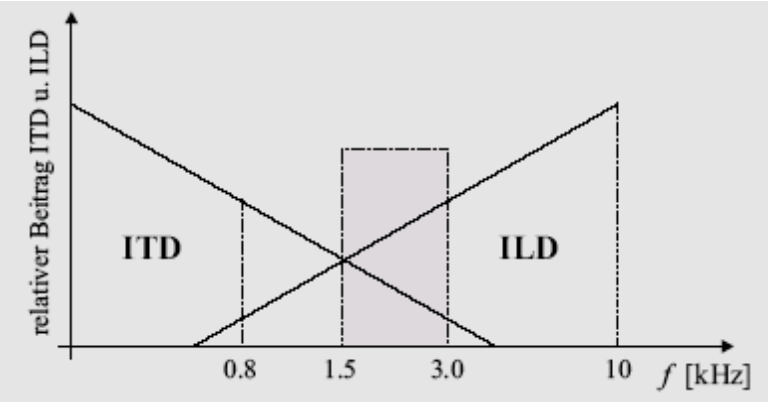


Abbildung 3.7: Frequenzabhängiger Beitrag von ITD und ILD („Duplex-Theorie“) ([Art-diss])

3.6.3 Andere Einflussgrößen der Lokalisation in der Horizontalebene

Die Lokalisationsfähigkeit des Menschen in der Horizontalebene wird nicht nur durch die Signalparameter ILD und ITD beeinflusst, sondern auch zu einem bestimmten Maße durch folgende Untersuchungsbedingungen:

- Signalfrequenz und –bandbreite
- Signaldauer
- Peilbewegungen des Kopfes
- Blickrichtung → Augenposition

Allgemein können breitbandige Signale mit einer niedrigeren Lokalisationsunschärfe als schmalbandige wahrgenommen werden. Die Lokalisation von Dauertönen und anderen Schmalbandsignalen weicht von derjenigen für Breitbandsignale ab[Bla83]. Dies soll die Abbildung 3.8 verdeutlichen. Sie zeigt die Auslenkung in Abhängigkeit der Frequenz, die ein Dauertöne bzw. Gauß-Töne abstrahlender Lautsprecher gegenüber einem Breitbandrauschen abstrahlenden Lautsprecher aufweisen muss, damit die Hörereignisse sich richtungsmäßig decken. Es sind drei Fälle dargestellt, die zu Seitenwinkel des rauschenden Lautsprechers von φ=320°, 0° und 40° gehören.

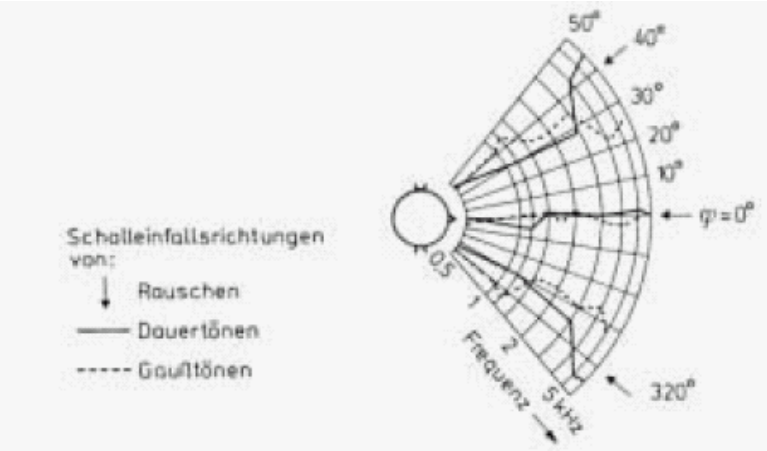


Abbildung 3.8: Lokalisation in der Horizontalebene in Abhängigkeit spektraler Signalparameter für 3 Richtungen φ= {0°, 40°, 320°} ([Bla83])

Des weiteren führt eine Erhöhung der Signaldauer bis 700 ms ebenfalls zu einer Verringerung der Lokalisationsunschärfe, während eine weitere Erhöhung der Signaldauer keine Verbesserung mehr bringt [Bla83].

Darüber hinaus spielen geringfügige Peilbewegungen des Kopfes, die immer wieder unbewusst einsetzen ebenfalls eine Rolle und können die Lokalisation erheblich erleichtern.

3.6.4 Hören in der Medianebene

Nun komme ich auf die Mechanismen zu sprechen, die für die Lokalisation in der Medianebene zuständig sind.

Das räumliche Hören in der Medianebene funktioniert ganz anders als das räumliche Hören in der Horizontalebene. Dies hängt damit zusammen, dass sich bei Bewegungen der Schallquelle innerhalb der Medianebene immer konstante interaurale Merkmale ergeben. Deshalb kann das Gehör hier mit Hilfe der interauralen Merkmale nicht die Richtung der Schallquelle bestimmen. Dennoch gelingt unserem Gehör innerhalb der Medianebene die Lokalisation, wenn auch mit einer größeren Lokalisationsunschärfe. Die Lokalisation innerhalb der Medianebene gelingt unserem Gehör mit Hilfe der so genannten *monauralen Merkmale* die in der Fachliteratur auch als „spectral cues“ bezeichnet werden.

Mit monauralen Merkmalen bezeichnet man die Fähigkeit des Gehörs, richtungsabhängige spektrale Eigenschaften des Gehörs auszuwerten, die vor allem durch die akustische Filterfunktion des Außenohres verursacht werden. Die Ohrmuschel und der Gehörgang des Menschen wirken nämlich als richtungsselektive Filter. In der Ohrmuschel werden je nach Einfallrichtung des Schalls aus der Medianebene unterschiedliche Resonanzen angeregt. Dabei überlagert sich direkt einfallender Schall mit in den Hohlräumen des Ohrs reflektiertem Schall. Damit besitzt jede Richtung aus der Medianebene ein unterschiedliches Resonanzmuster. Das heißt, dass unsere Ohren einen richtungsabhängigen Frequenzgang besitzen, den unser Gehirn-

Gehör-System auswerten kann. Es sei hier noch angemerkt, dass im Gegensatz zu interauralen Merkmalen, die monauralen Merkmale auch mit nur einem Ohr ausgewertet werden können.

Die so genannten Außenohrverzerrungen spielen eine zentrale Rolle beim räumlichen Hören in der Medianebene und sind eine der wesentlichen monauralen Merkmale. Mit Hilfe folgender Abbildung möchte ich auf die Außenohrverzerrungen näher eingehen:

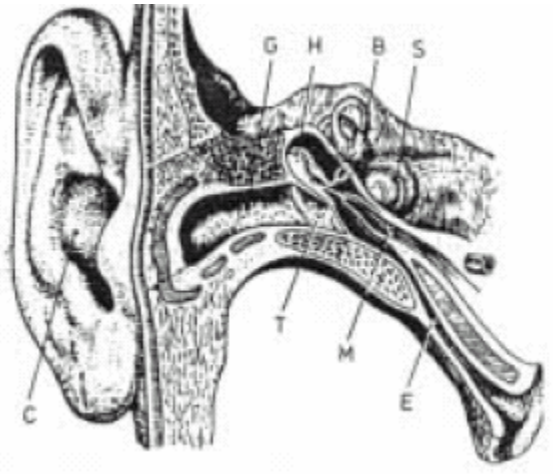


Abbildung 3.9: Ohrenmodell (nach [Bla83]);
Legende: B: Bogengänge, S: Schnecke, M: Trommelfellspanner, C: Cavum conchae (Haupthöhle), G: äußerer Gehörgang, T: Trommelfell, H: Hammer

Wie man an der Abbildung 3.9 sieht, besteht das Außenohr aus der Ohrmuschel und dem Eingang des Gehörganges. Die Ohrmuschel besteht dabei aus Vertiefungen und Erhebungen und weist zahlreiche Höhlen auf, in denen sich der Schall sammeln kann. Der untere Teil der Ohrmuschel besteht aus dem Ohrläppchen, und fungiert als Resonanzkörper. Diese Höhlen, Erhebungen und Vertiefungen stellen ein Filtersystem dar, welches je nach Schalleinfallsrichtung unterschiedlich angeregt wird. Dabei spielen physikalische Effekte wie Reflexion, Abschattung, Streuung, Beugung, Interferenz eine Rolle. Diese Effekte finden nicht nur am Außenohr statt, sondern auch in einem geringeren Maße am Rumpf und Kopf. Je nach Schalleinfallsrichtung findet somit eine individuelle Klangverfärbung statt, die dazu genutzt wird, zwischen vorne, hinten, oben und unten zu unterscheiden. Es ist noch wichtig, anzumerken, dass das Auftreten dieser Effekte abhängig vom Verhältnis der Wellenlänge des Schalls zur jeweiligen Grenzfläche (in diesem Fall der Ohrmuschel) ist. Wenn man die Größe der Ohrmuschel in Betracht zieht und diese mit den unterschiedlichen Wellenlängen des Schalls vergleicht, wird klar, dass spektrale Veränderungen vorwiegend bei Frequenzen $f \geq 4$ kHz [1] relevant sind. Die Wirksamkeit monauraler Merkmale setzt also eine breitbandige Erregung mit hochfrequenten Anteilen voraus.

[1] Das ist der Grund, warum das Gehör Bass nicht lokalisieren kann, weil es aus sehr tiefen Frequenzen besteht.

Die Frequenzabhängigkeit der monauralen Merkmale kann durch die *monaurale (komplexe) Übertragungsfunktion* des äußeren Ohres beschrieben werden. Dies ist die Übertragungsfunktion zwischen dem Schalldruck an einem Messpunkt im Gehörgang bei einer beliebigen Schalleinfallsrichtung und Schallquellenentfernung, bezogen auf den Schalldruck am gleichen Messpunkt bei Schalleinfall von einer Bezugsschallquelle in einer Bezugsrichtung und Bezugsentfernung. In der Regel wird eine ebene Welle aus $\varphi=0°$, $\delta=0°$ als Bezugsschall angenommen. Es sei hier noch anzumerken, dass gesondert angegeben werden muss, für welche Schallquelle, welche Entfernung und Richtung sie jeweils gemessen wurde. Eine individuelle Messung muss also für jede Schallquellenposition separat erfolgen. Man muss sich hier im Klaren darüber sein, dass unsere Ohrmuscheln nicht für jede Richtung gleich filtern, sondern richtungsabhängig filtern. Deshalb sind die Übertragungsfunktionen auch von φ und δ abhängig. Ich werde später, wenn ich auf die *Kopfbezogenen Übertragungsfunktionen (HRTF)* eingehe, näher auf die eben geschilderte Problematik eingehen.

Eine weitere Überlegung, die verdeutlicht, dass interaurale Merkmale alleine für das Richtungshören nicht verantwortlich sein können, ist die Tatsache, dass sie Mehrdeutig sind. So lassen sich in der Horizontalebene Orte mit konstanten interauralen Merkmalen finden, die auf Bahnen liegen, die wie eine Hyperbel aussieht. Auf diesen Hyperbeln haben alle Punkte zu den beiden Ohrenpunkten die gleiche Abstandsdifferenz. Schallwellen, die von irgendeinem Punkt dieser Hyperbel ausgehen, erreichen die beiden Ohren mit einer konstanten interauralen Zeitdifferenz. Schaut man sich diese Bahnen aus dem drei dimensionalen Raum an und nähert man die Hyperbeln durch ihre Asymptoten an, so erhält man einen geometrischen Ort, der so aussieht, wie die Mantelfläche eines Kegels. Diese Kegel werden in der Fachliteratur auch als *Kegel der Verwechselung („cones of confusion")* bezeichnet. Die Abbildung 3.10 zeigt diese Kegel der Verwechselung:

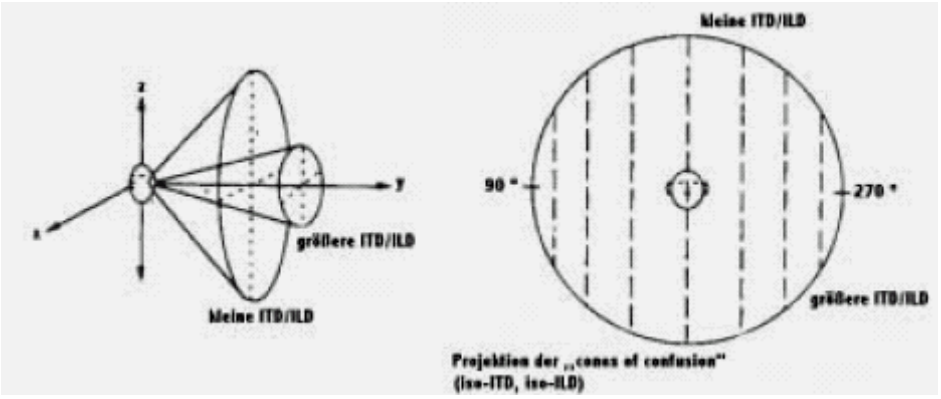


Abbildung 3.10: Orte mit konstanten interauralen Merkmalen (ITD und ILD) („cones of confusion") ([WADW93])

Der rechte Teil der obigen Abbildung zeigt Linien, die durch die Projektion der Kegel auf die Horizontalebene entstehen. Verläuft die Bahn einer Schallquelle entlang dieser Linien und strahlt diese dabei schmalbandige Signale ab, so nimmt auch die Wirksamkeit der Monauralen Merkmale ab und es entstehen so genannte „front-back confusions".

Zum Schluss möchte ich noch darauf hinwiesen, dass interaurale und monaurale Merkmale nie getrennt voneinander auftreten, sondern immer beide gemeinsam vorhanden sind. Monaurale Merkmale erfüllen außerdem noch die Funktion, dass sie interaurale Zweideutigkeiten reduzieren können, und somit das Auftreten von Lokalisationsfehlern, wie „front-back-confusions“ verhindern können.

3.6.5 Einflussgrößen der Lokalisation in der Medianebene

Die Lokalisationsfähigkeit des Menschen in der Medianebene ist von folgenden Signalparametern und Untersuchungsbedingungen abhängig:

- Signalfrequenz- und –bandbreite
- Signaldauer
- Signalkenntnis/ -vertrautheit
- Peilbewegungen des Kopfes
- Blickrichtung → Augenposition

Die Lokalisation in der Medianebene ist nur für Signale mit einer Bandbreite von $\Delta f \geq$ 1-2 Terzen möglich. Für den Fall schmalbandigerer Signale lassen sich für die Medianebene weder Lokalisationen noch Lokalisationsunschärfen bestimmen. Bei solchen Signalen hängt die Hörereignisrichtung nicht von der Schallquellenrichtung, sondern nur von der Signalfrequenz ab. Die Zuordnungsvorschrift Schallquellenrichtung-Hörereignisrichtung ist also unbestimmt [Bla83]. In der Abbildung 3.11 ist die Bahn beschrieben, die das Hörereignis beschreibt, wenn ein Lautsprecher aus irgendeiner beliebigen Richtung der Medianebene ein Schmalbandrauschen variabler Mittenfrequenz abstrahlt.

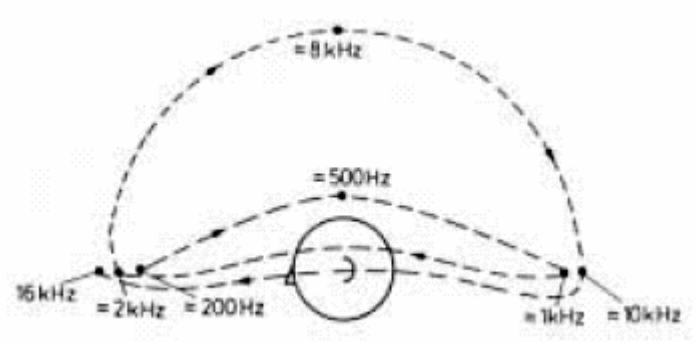


Abbildung 3.11: Hörereignisbahnen in Abhängigkeit von der Mittenfrequenz bei Schmalbandrauschen $\Delta f \leq$1-2 Terzen ([Bla83])

Das Hören in der Medianebene ist ein lernabhängiger Vorgang. So zeigen Untersuchungen, dass das Lokalisieren von bekannten Geräuschen zu einer niedrigeren Lokalisationsunschärfe in der Medianebene führt (ausreichende Bandbreite sei hier vorausgesetzt, damit nicht der oben geschilderte Effekt eintritt).

Auf den Einfluss von Kopfbewegungen und Augenposition möchte ich hier nicht näher eingehen, da diese Einflussgrößen bei der Simulation mittels Kopfhörers nicht mit simuliert werden können. Da es das Ziel dieser Arbeit ist, die Wirksamkeit und Grenzen einer Kopfhörersimulation für eine Anwendung für Blinde Personen auf dem Computer (GUIB-Projekt) zu untersuchen, sind diese Einflussgrößen irrelevant.

Zu dem Einfluss der Augenposition sei nur kurz gesagt, dass diese für die akustische Position insofern von Bedeutung sind, dass das bessere Lokalisationsvermögen der Augen für die akustische Lokalisation als Kalibrierung dienen kann.

3.7 Systemtheorie der binauralen Simulation

In diesem Abschnitt möchte ich beschreiben, wie man mit Hilfe von Kopfhörern einen räumlichen Höreindruck erzielen kann.

Die Idee dabei ist, mit den Kopfhörern am Trommelfell die gleichen Signalverhältnisse zu erzeugen, wie sie auch unter realen Freifeldbedingungen vorliegen würden. Wenn dies gelingt, müsste für den Hörer der gleiche räumliche Höreindruck entstehen, wie unter natürlichen Bedingungen auch, da am Trommelfell alle Richtungsinformationen bereits im Signal kodiert sind. Die Signale am Trommelfell würden in diesem Falle also den gleichen Informationsgehalt besitzen, wie auch unter Freifeldbedingungen. Es muss noch darauf hingewiesen werden, dass sich nur physikalische Eigenschaften simulieren lassen, nicht aber psychoakustische Abhängigkeiten (Einfluss von Kopfbewegungen, Augenposition → Blickrichtung etc.).

3.7.1 Räumliches Hören als LZI-System

Man kann das räumliche Hören als lineares, zeitinvariantes Übertragungssystem (LZI-System) ausfassen. Die Abbildung 3.12 zeigt ein Blockschaltbild dieses Systems:

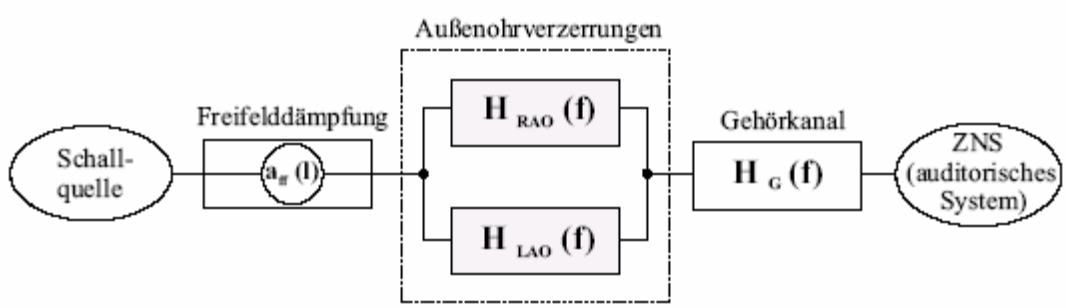


Abbildung 3.12: Räumliches Hören als LZI-System ([Art-diss])

Das System kann deshalb als linear bezeichnet werden, da das Eingangssignal dem Ausgangssignal immer proportional ist. Außerdem ändert das System seine inneren Eigenschaften nicht und reagiert auf Eingangssignale immer wieder gleich, unabhängig vom Zeitpunkt ihres Eintreffens (daher Zeitinvariant).

Wie man an der obigen Abbildung sieht, ist die Schallübertragung in 3 Bereiche unterteilt:

1.) Freifeldübertragung (von der Schallquelle bis zur Ohrmuschel)
2.) Außenohrübertragung (von den beiden Ohrmuscheln bis zum Gehöreingang[2])
3.) Übertragung im Gehörgang (vom Gehöreingang bis zum Trommelfell)

Wie man an der Abbildung sieht, ist die *Freifeldübertragung* lediglich eine Dämpfung. Bis etwa 15 m wird der Schalldruckverlauf durch die beiden linearen Schallfeldgleichungen beschrieben. Also ist in diesem Bereich die Übertragungsstrecke verzerrungsfrei. Erst bei einer größeren Entfernung ist die Übertragung nicht mehr verzerrungsfrei, da bei größeren Entfernungen die Dämpfung frequenzabhängig[3] wird. In jedem Fall ist die Dämpfung aber *richtungsunabhängig* und enthält somit keine Richtungsinformationen.

Bei der *Außenohrübertragung* findet eine Filterung durch die beiden Ohrmuscheln statt. Diese Filterung führt zu einer linearen Verzerrung, die richtungs- und entfernungsabhängig ist. Diese richtungsabhängigen Verzerrungen werden durch die beiden Außenohrübertragungsfunktionen Hlao(f,φ,δ) und Hrao(f, φ, δ) beschrieben.

Schließlich gelangt der Schall in den Gehörgang. Der Gehörgang kann mit Hilfe eines elektrischen Ersatzschaltbildes beschrieben werden. Wenn man sich den Gehörgang anschaut, sieht man, dass er aus einem Kanal besteht, dessen Wände den Schall kaum absorbieren. Daher kann der Gehörgang mit einer schwach verlustbehafteten Leitung verglichen werden, die einen akustischen Wellenwiderstand Zw besitzt und am Ende mit der Trommelfellimpedanz Ztr abgeschlossen ist. Die Abbildung 3.13 zeigt in Analogie zur Leitungstheorie das Ersatzschaltbild.

[2] Näherungsweise ab einer Eintrittstiefe von ca. 5 mm in den Gehörgang
[3] hohe Spektralanteile werden stärker gedämpft als niedrige

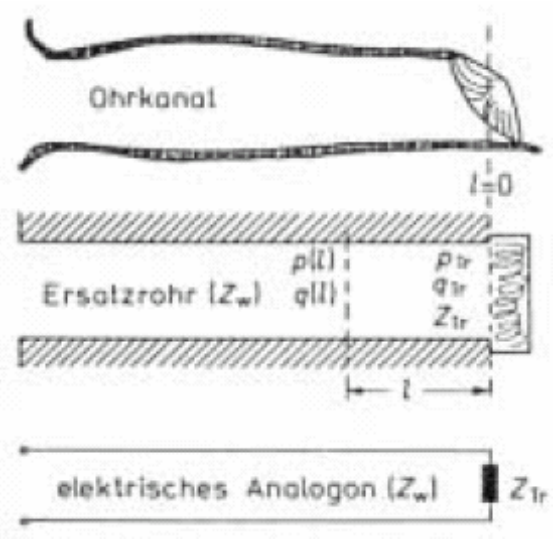


Abbildung 3.13: Ersatzschaltbild der Schallausbreitung im Gehörgang ([Art-diss])

Folgende Gleichungen beschreiben dabei den komplexen Schalldruck- bzw. Schallschnelleverlauf, wenn der Gehörkanal sinusförmig erregt wird:

$$p(l) = p_{Tr} \cosh \gamma l \; + \; \frac{Z_W}{Z_{Tr}} p_{Tr} \sinh \gamma l, \tag{3.4}$$

$$q(l) = q_{Tr} \cosh \gamma l \; + \; \frac{Z_{Tr}}{Z_W} p_{Tr} \sinh \gamma l \tag{3.5}$$

Dabei ist der Schalldruck p(l) die zur die zur elektrischen Spannung und die Schallschnelle q(l) die zum elektrischen Strom analoge Größe.

Der Schalldruck kann genau so, wie in der Leitungstheorie auch, von einer Stelle zur Anderen transformiert werden. Für die Schalldruckübertragungsfunktion an einem beliebigen Ort im Gehörgang gilt folgende Gleichung:

$$H_G(f) = \frac{p_{Tr}}{p(l)} = \frac{1}{\cosh \gamma l \; + \; \frac{Z_W}{Z_{Tr}} \sinh \gamma l} \tag{3.6}$$

3.7.2 Die Kopfbezogene Übertragungsfunktion HRTF (Head-Related Transfer Functions)

Wenn man die 3 Übertragungsfunktionen der einzelnen Abschnitte addiert, bekommt man eine Übertragungsfunktion des Gesamtsystems. Diese Übertragungsfunktion wird in der Psychoakustik als HRTF (head-related-transfer-function) bezeichnet.

Was an der Abbildung des LZI-Systems nicht ganz klar wird, ist die Tatsache, dass das linke und rechte Ohr getrennt betrachtet werden müssen. Es gibt also zwei verschiedene LZI-Systeme und somit auch zwei verschiedene HRTF ($HRTF_L$ und $HRTF_R$).

Sind nun für eine bestimmte Position im Raum die Übertragungsfunktionen des rechten und linken Ohres bekannt, so lassen sich für ein beliebiges Eingangssignal aus dieser Position die beiden Ausgangssignale an den beiden Trommelfellen errechnen.

Mathematisch ausgedrückt würde man das Ausgangssignal folgendermaßen bekommen:

$$Y_L(f) = HRTF_L(f)\, X_L(f) \tag{3.7}$$

$$Y_R(f) = HRTF_R(f)\, X_R(f) \tag{3.8}$$

Dabei sind X und Y die Fouriertransformierten den Eingangs und Ausgangssignals. Man kann also die HRTF für eine bestimmte Position im Raum bestimmen, wenn man das Ausgangssignal am Trommelfell misst, es dann fouriertransformiert und durch die Fouriertransformierte des Eingangssignals teilt. Dabei darf die Fouriertransformierte des Eingangssignals im interessierenden Bereich keine Nullstellen aufweisen, da die HRTF sonst an dieser Stelle nicht definiert wären.

Ich möchte nochmals darauf hinweisen, dass die HRTF richtungsabhängig sind. Daher existiert für jede Position im Raum ein Paar von Übertragungsfunktionen ($HRTF_L(f,\varphi,\delta)$ und $HRTF_R(f,\varphi,\delta)$).

Es bietet sich hier natürlich auch die Möglichkeit an, im Zeitbereich zu rechnen. Dafür braucht man die Impulsantwort des Systems. Diese Erhält man, wenn man die inverse Fouriertransformierte der HRTF bildet. Die Impulsantwort wird in der Psychoakustik **HRIR** (**H**ead-**R**elated-**I**mpuls-**R**esponse) genannt. Mit Hilfe des Faltungsintegral lassen sich dann für beliebige Eingangssignale die Ausgangssignale des Systems errechnen. Die Beachtron-Karte, die ich für meine Untersuchungen verwendet habe, verwendet das Faltungsintegral, um die Ausgangssignale des Systems zu errechnen. In der Datenbank der Treibersoftware dieser Karte sind also nicht die HRTF gespeichert, sondern die HRIR. Ausführliche Details über die Beachtron-Karte werde ich später liefern.

3.7.3 Messung der HRTF

Nun möchte ich beschreiben, wie die HRTF in der Praxis gemessen werden können und welche messtechnischen Probleme dabei gelöst werden müssen.

Die Abbildung 3.14 zeigt einen messtechnischen Aufbau zur Bestimmung der HRTF.

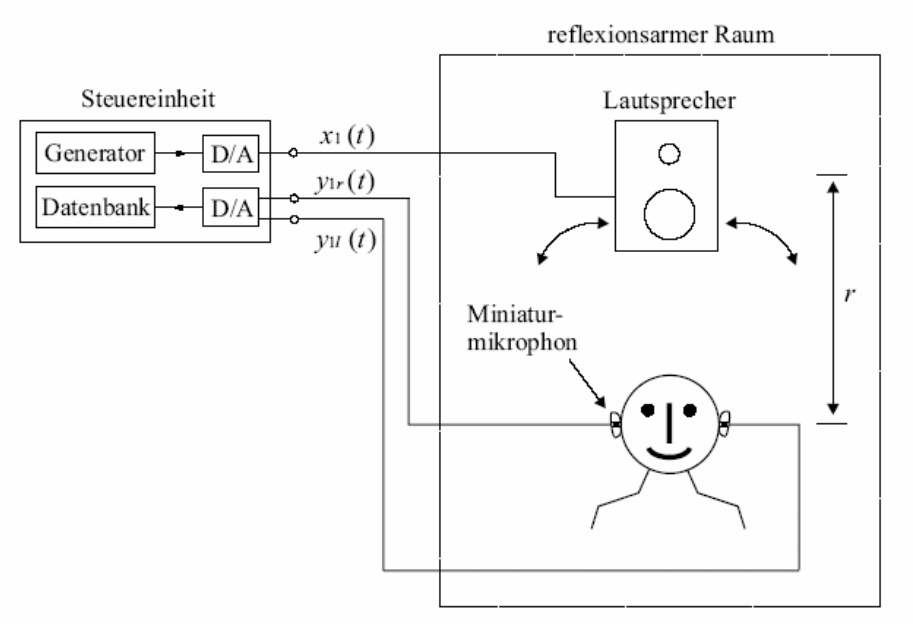


Abbildung 3.14: Messaufbau zur Bestimmung der HRTF ([Art-diss])

In der obigen Abbildung wird die Lautsprechermembran eines Lautsprechers mit Hilfe einer Signalprozessorkarte impulsförmig angeregt. Die gesamte Anordnung befindet sich zur Minimierung von Störeinflüssen in einem reflexionsarmen Raum. Der Lautsprecher ist um den Kopf herum drehbar, sodass für alle Positionen die Impulsantwort bestimmt werden kann. Die Impulsantwort wird mit Hilfe von Miniaturmikrophonen in der Nähe des Trommelfells aufgezeichnet. Die Miniaturmikrophone und die Leitungen, die zu ihnen führen, müssen möglichst klein gehalten werden, um das Schallfeld nicht zu stören.

Die so durch aufgezeichnete Antwort des Systems enthält neben der HRIR auch noch die Übertragungsfunktion des Lautsprechers und des Mikrofons. In der Messkette treten also ungewünschte Verzerrungen auf. Im Frequenzbereich lässt sich das folgendermaßen ausdrücken:

$$Y_1(f) = X_1(f)\, H_{LS}(f)\, HRTF(f)\, H_M(f) \qquad (3.9)$$

Wenn man mit Hilfe der HRTF später eine Kopfhörersimulation durchführt, treten zusätzliche Verzerrungen durch die Übertragungsfunktion des Kopfhörers auf. Wenn aber die

Übertragungsfunktion des Kopfhörers bekannt ist, kann mittels inverser Filterung diese Verzerrung aufgehoben werden. Die Übertragungsfunktionen des Lautsprechers und des Mikrophons lassen sich mit Hilfe von einer Referenzmessung eliminieren. Die Abbildung 3.15 zeigt, wie eine solche Referenzmessung durchgeführt werden kann.

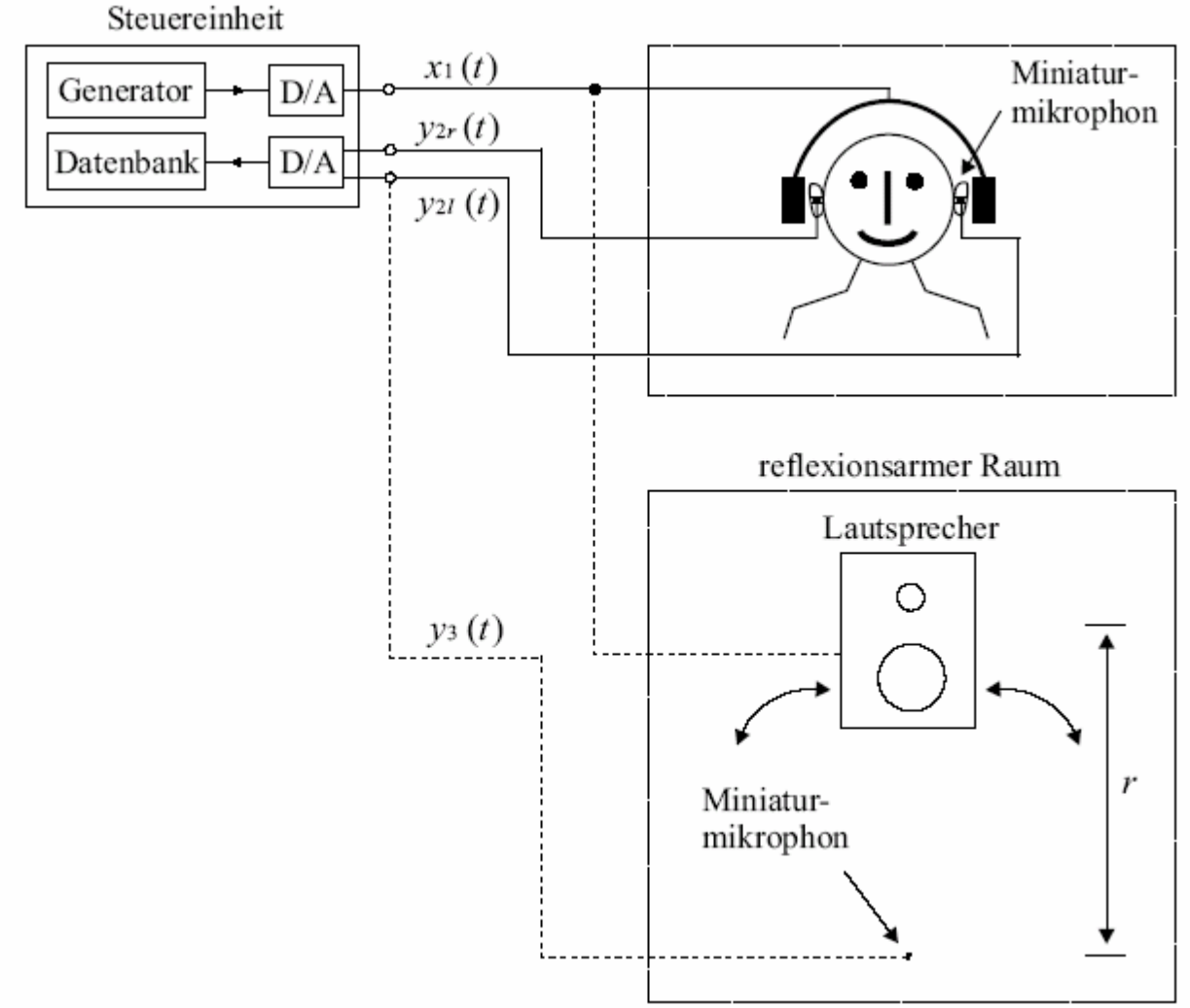


Abbildung 3.15: Messaufbau zur Bestimmung von Kopfhörer- und Lautsprecherübertragungsfunktion (Referenzmessungen) ([Art-diss])

Der untere Teil der Abbildung 3.15 zeigt die Referenzmessung zur Eliminierung der Übertragungsfunktion des Lautsprechers und des Mikrophons. Dabei platziert man das Miniaturmikrophon an die gleiche Stelle im Raum, wo sich später der Kopf der Versuchsperson befinden würde. Erregt man nun den Lautsprecher impulsförmig und misst die Antwort $y_3(t)$, bekommt man im Frequenzbereich folgende Gleichung:

$$Y_3(f) = X_1(f)\, H_{LS}(f)\, H_M(f) \qquad (3.10)$$

Teilt man die vorher gewonnene Gleichung (als der Kopf der Versuchsperson noch in der Messkette drin war), durch die obige Gleichung, kürzt sich alles raus, bis auf die HRTF:

$$Y_1(f)/Y_3(f) = HRTF(f) \qquad (3.11)$$

Damit ist ein Weg gefunden, die HRTF für beliebige Raumpositionen zu messen. Dabei muss die Referenzmessung natürlich nur einmal durchgeführt werden.

In der Praxis werden die HRTF in der Regel mit Hilfe von Kunstköpfen gemessen. Darüber hinaus werden die HRTF nicht nur mit Hilfe der Impulsantworten sondern oft auch mit Rauschsignalen gemessen.

Abschließend möchte ich noch auf eine Besonderheit der binauralen Simulation aufmerksam machen: Da die Impulsantworten unmittelbar vor dem Trommelfell gemessen werden, müsste man bei der Wiedergabe der Signale mittels Kopfhörer die Kopfhörer Idealerweise auch unmittelbar vor dem Trommelfell platzieren. Da aber Miniaturkopfhörer extrem teuer wären, verzichtet man auf diese Lösung und setzt die Kopfhörer direkt auf die Ohrmuschel auf. Dadurch wird das Signal aber ein zweites Mal durch die Ohrmuschel und den Gehörgang verzerrt. Die Verzerrung durch die Ohrmuschel ist in diesem Falle aber vernachlässigbar, da sich aus dieser Nähe kein *ebenes Wellenfeld* aufbauen kann. Die Ohrmuschel kann ihre volle Filterwirkung aber nur entfalten, wenn eine annähernd ebene Welle aus genügend großer Entfernung auf sie trifft. Am Ende bleibt also nur noch die wiederholte Verzerrung des Gehörgangs als störender Rest übrig.

4 Psychoakustische Messungen auf dem Gebiet der menschlichen Lokalisationsschärfe

4.1 Einleitung

In diesem zweiten Abschnitt möchte ich ausführlich auf meine eigene Aufgabenstellung eingehen und beschreiben, was ich während der Diplomarbeit gemacht habe.

Meine Aufgabe bestand darin, die Fähigkeit eines Systems für Kopfhörerwiedergabe zu untersuchen. Sinn und Ziel der Untersuchung war es, herauszufinden, wie gut sich eine Kopfhörersimulation für das GUIB-Projekt eignet. Wie ich bereits in der Einleitung dieser Arbeit erwähnte, versucht das GUIB-Projekt, blinden Personen die Arbeit mit dem Computer zu erleichtern. Die Abkürzung GUIB steht dabei für ***G****raphical* ***U****ser* ***I****nterface for* ***B****lind* ***P****ersons*.

Die Idee hinter der Sache ist es, Bildschirm-Icons und Bildschirmereignisse (wie das Öffnen und Schließen eines Fensters, Tastatureingaben, Bewegungen des Mauszeigers uvm.) durch so genannte „Earcons“ (auditive Repräsentation von Bildschirm-Icons) zu ersetzten. Man versucht also, blinden Personen, denen eine visuelle Lokalisation fehlt, mittels einer akustischen Lokalisation eine Orientierung auf dem Bildschirm zu ermöglichen.

Im Prinzip kann man virtuelle Schallereignisse auch mit Hilfe von Lautsprechern (z.B. Standard-Stereophonie oder Mehrkanal-Lautsprecheranordnungen) realisieren. Diese haben aber den Nachteil, dass sie viel Platz einnehmen und in einer bestimmten Anordnung im Raum platziert werden müssten. Der Vorteil bei der Lautsprecherwiedergabe wäre aber, dass in diesem Falle das Ohr seinen natürlichen Richtungsfiltereffekt ausnutzen kann. Typische Fehler wie Vorn-Hinten-Lokalisation, axialsymmetrische Richtungsänderungen und Im-Kopf-Lokalisation, wie sie bei Kopfhörersimulation häufig auftreten, wären bei Lautsprecherwiedergabe nicht vorhanden. Trotzdem sind der große Volumen von mehreren Lautsprechern und der dadurch entstehende Platzbedarf, sowie die höheren Kosten gegenüber einer Kopfhörersimulation ein entscheidender Nachteil, weswegen die Simulation mittels Kopfhörern bevorzugt wurde.

4.2 Grundbegriffe und Definitionen

4.2.1 VAD (Virtuell Audio Display)

Allgemeine Definition von VAD:

VAD bedeutet, mit virtuellen Schallquellen eine solche Umgebung zu erzeugen, dass der Hörer die Schallereignisse lokalisieren, identifizieren und diskriminieren kann. Die Schallereignisse,

als akustische Darstellungen können auch als ergänzende Information neben den visuellen erscheinen (mit Bildschirm). VADs – man nennt sie auch Spatial Auditory Displays - können sowohl blinde Rechnerbenutzer als auch Flugsimulatorpiloten oder Computerspieler verwenden.

Da jeder Computerbildschirm eine 2 dimensionale und flache Oberfläche hat, wurde für das GUIB-Projekt ebenfalls ein 2D VAD zugrunde gelegt. Die Abbildung 4.1 zeigt das 2D VAD.

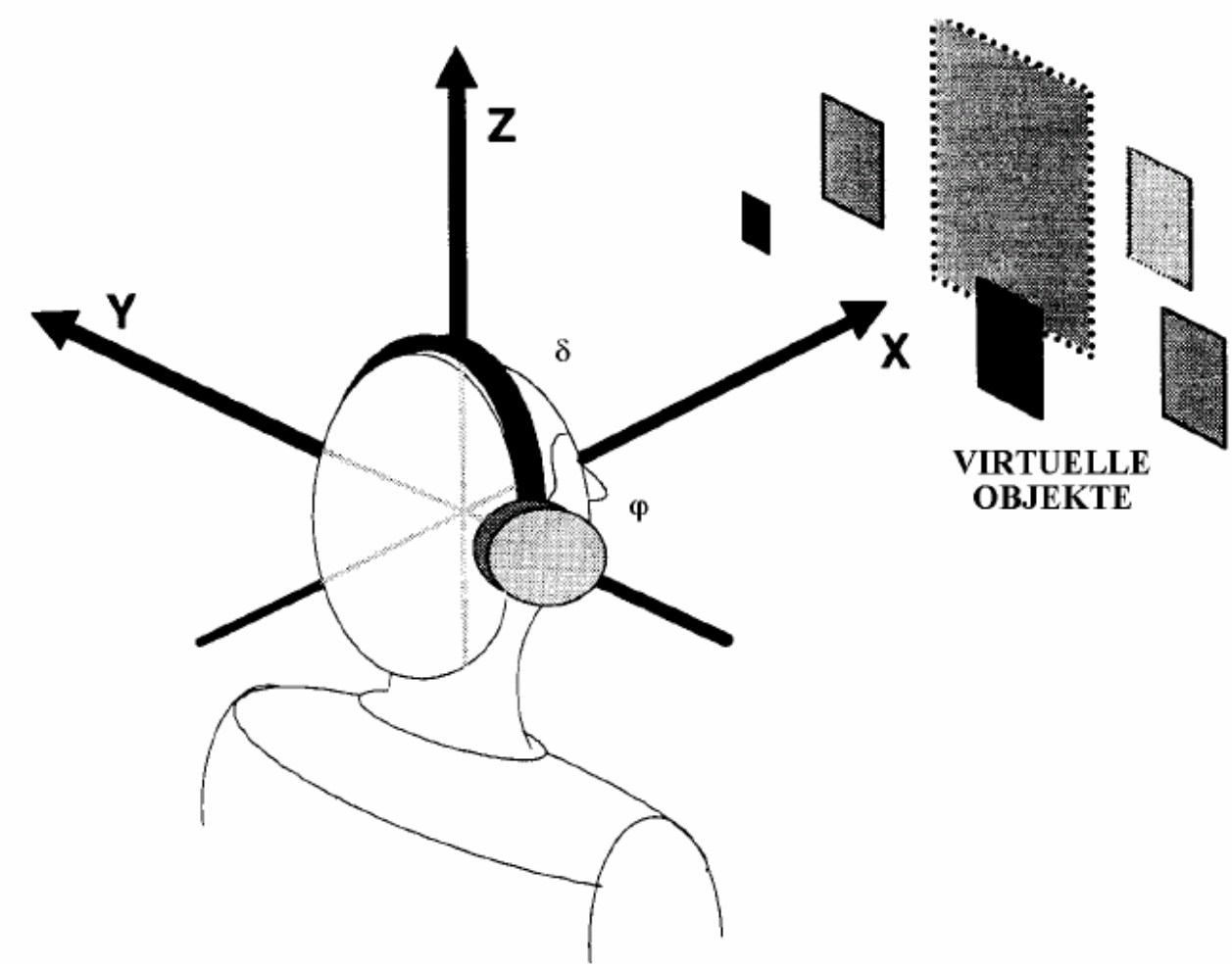
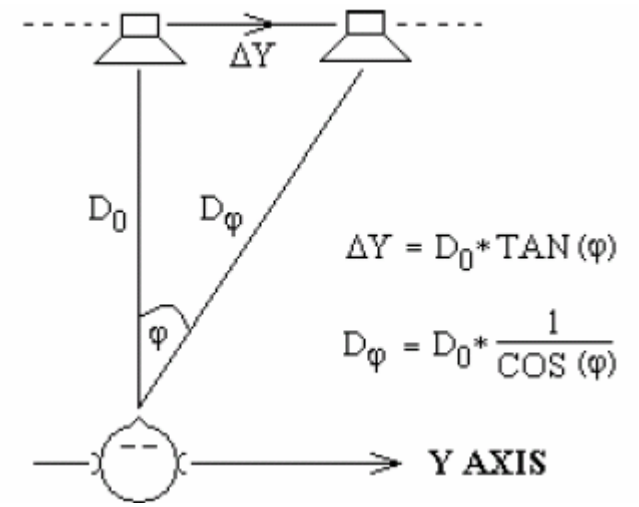


Abbildung 4.1: 2D akustische Darstellung. Die akustische Oberfläche ist parallel mit der Y-Z-Ebene. Ihr Mittelpunkt (Ursprung) ist in der Vorwärtsrichtung, $\varphi=\delta=0$. Virtuelle Schallquellen während der Untersuchung bewegen sich entweder parallel mit der Y-Achse, oder mit der Z-Achse. ([Wers-diss])

Man sieht in der Abbildung 4.1 eine 2D akustische Darstellung. Die akustische Oberfläche ist parallel mit der Y-Z-Ebene. Ihr Mittelpunkt (Ursprung) ist in der Vorwärtsrichtung, $\varphi=\delta=0$. Virtuelle Schallquellen während der Untersuchung bewegen sich also entweder parallel mit der Y-Achse, oder mit der Z-Achse. Bei dem obigen Koordinatensystem handelt es sich um ein Kopfbezogenes Koordinatensystem (Head-Related Coordinate System), das in dem ersten Abschnitt dieser Arbeit schon beschrieben wurde.
Man hätte natürlich auch eine kugelförmige oder zylinderförmige Oberfläche wählen können. Da eine solche Zuordnungen jedoch nicht den üblichen Anwendungen entsprochen hätten, wie z.B. unter MS Windows, wo sämtliche Ereignisse, Fenster und Icons auf einer rechteckförmigen Oberfläche erscheinen, hat man für das GUIB-Projekt auch eine rechteckförmige Oberfläche gewählt. Die 2D Oberfläche hat dabei einen maximalen Winkel von ± 60° seitlich und vertikal und ist 50 Zoll (127 cm) vom Koordinatenursprung entfernt.
Ich bin der Ansicht (und auch diejenigen, die damals das Guib Projekt ins leben gerufen haben), dass die Interpretation zwischen Bildschirm und Mausbewegungen besser und leichter sind, wenn dieses VAD beschränkt ist (z.B. 60 Grad) und auf keinen Fall erreicht 90° erreicht oder sich hinter dem Kopf fortsetzt. Würde man sich zum Beispiel für eine Kugelförmige Oberfläche entscheiden, würden sich folgende technische Schwierigkeiten ergeben:

- die Maus liegt auf einer 2 dimensionalen Oberfläche, die virtuelle kugelförmige Oberfläche ist jedoch 3 dimensional → es wird schwierig, die 2 dimensionalen Bewegungen der Maus auf eine 3 dimensionale Kugeloberfläche zu übertragen
- Maus und Touchpads sind oft so empfindlich, dass kleine Bewegungen zu überproportionalen Zeigerbewegungen führen

Wenn sich die virtuelle Schallquelle innerhalb des 2D VADs seitlich oder vertikal bewegt, nimmt mit zunehmendem Winkel δ und φ auch der Abstand der Schallquelle zum Koordinatenursprung zu. Die geringste Entfernung zur Versuchsperson ist dabei in der Vorwärtsrichtung (D0). Mit steigender Auslenkung steigt die Entfernung mit einer 1/COS-Funktion des Auslenkungswinkels δ oder φ. Abbildung 4.2 veranschaulicht dies.



$$\Delta Y = D_0 * \mathrm{TAN}(\varphi) \qquad (4.1)$$

$$D_\varphi = D_0 * \frac{1}{\mathrm{COS}(\varphi)} \qquad (4.2)$$

Abbildung 4.2: Die Schallquellen-Entfernung in der Horizontalebene ist eine 1/COS-Funktion des Seitenwinkels φ, wenn die Schallquellen sich parallel mit der Y-Achse bewegen. Die Y-Koordinaten sind eine TAN-Funktion des Seitenwinkels. ([Wers-diss])

Analog verhält sich die Zunahme der Entfernung auch bei Bewegungen parallel zur Z-Achse. Da die Beachtron-Karte ein kartesisches Koordinatensystem benutzt, unsere Angaben aber während der Untersuchung immer auf 2 variablen Winkelangaben (δ und φ) und einer festen Entfernungsangabe (Entfernung der 2D VAD Oberfläche zum Koordinatenursprung →x-Koordinate) beruhten, musste ich innerhalb der beiden Programme die Winkel und Entfernungsangabe mittels trigonometrischer Beziehungen umrechnen.

4.2.2 Untersuchungsmethoden

Prinzipiell gibt es drei verschiede Möglichkeiten, die menschliche Lokalisationsfähigkeit zu untersuchen:

- Absolute Messung
- MAA Messung (Minimum Audible Angle)
- Intervall-Urteilmethoden

Bei der **„absoluten“** Messung muss die Versuchsperson den genauen Ort der Schallquelle angeben. Dabei hat sie Möglichkeit, entweder mit dem Finger oder mit einem Zeiger auf die Schallquelle zu zeigen oder aber die Koordinaten der Schallquelle nennen.

Bei der so genanten **MAA** Messung muss die Versuchsperson nur auf die Änderung der Schallquellenrichtung achten. Bei einer solchen Untersuchung sind zwei Schallquellen vorhanden. Beide davon befinden sich zunächst am selben Ort und geben abwechselnd voneinander Signale ab. Dann fängt die zweite Schallquelle an sich entweder nach recht, links, oben oder unten zu bewegen. Die andere Schallquelle, die so genannte Referenzschallquelle bleibt dabei solange stehen, bis die Versuchspeson eine Richtungsänderung der zweiten Schallquelle wahrnimmt. Sobald die Versuchsperson eine Richtungsänderung bemerkt und diese anzeigt, wird die Referenzschallquelle zum Ort der zweiten Schallquelle bewegt, nämlich zu dem Ort, wo vorher eine Richtungsänderung der zweiten Schallquelle wahrgenommen wurde. Dann wird das gleiche Spiel in die gleiche Richtung fortgesetzt, bis das Ende des 2D VAD erreicht ist. Mit einer solchen Methode kann man den gerade wahrnehmbaren Unterschied feststellen und zwar in Abhängigkeit des Schallquellenortes innerhalb der Median- und Horizontalebene.

Bei den so genanten **Intervallurteilmethoden** wird die Versuchsperson angewiesen, auf die Größe der Unterschiede zwischen dem Wahrgenommen zu achten. Diese Methode habe ich für meine Untersuchungen nicht benutzt.

4.3 Details zur Beachtron-Karte

Die Beachtron-Karte ist ein Produkt der Firma *Crystal River Engineering, Incorporation,* die in den USA (Kalifornien) ansässig ist. Es ist ein recht altes Produkt dieser Firma (aus dem Jahre 1993) und kann deshalb auch nur in älteren Rechnern problemlos eingesetzt werden (486 er). Die Beachtron Karte benutzt den ISA-Bus, um mit dem Rechner zu kommunizieren.
Mittlerweile bietet die Firma eine Reihe weiterer Produkte für die Kopfhörersimulation an, die in ihrer Leistungsfähigkeit und Qualität die Beachtron-Karte übersteigen.

Die Beachtron-Karte ist in der Lage, beliebige 16-bit Audio Dateien, die als Wave-Dateien auf der Festplatte gespeichert sind, mit Hilfe der HRTFs aus der eigenen Treiberdatenbank so umzurechnen und mittels eines geeigneten Kopfhörers so auszustrahlen, das ein räumlicher Höreindruck entsteht. Außerdem hat es einen externen stereo Eingang, sodass auch externe Audioquellen benutzt werden können. Insgesamt können 2 verschiedene Audioquellen simultan simuliert werden.

Für meine Untersuchungen haben wir einen Kopfhörer der Firma Sennheiser HD 540 benutzt, da die Beachtron-Karte laut Hersteller die Übertragungsfunktion dieses Kopfhörers kennt und somit in der Lage ist, mittels inverser Filterung die Verzerrung des Kopfhörers auf zu beheben.

Die HRTFs der Beachtron-Karte stammen aus einer Untersuchung von *Wightman und Kistler*. Dabei wurden die HRTFs einer weiblichen Versuchsperson aufgenommen, die besonders gut lokalisieren konnte. Es wurden jeweils für das linke und rechte Ohr insgesamt 72 HRTFs aufgenommen, die auf 6 verschieden Elevationen (zwischen –36° und +54°) verteilt waren und horizontal in 30° Schritten gemessen wurden. Pro Elevation wurden somit für jeweils das linke und rechte Ohr 12 HRTFs aufgenommen. Die fehlenden HRTFs sind von den vier benachbarten HRTFs interpoliert.

Da die Beachtron-Karte im Zeitbereich rechnet, wurden die HRIRs mit Hilfe von so genannten Minimalphasen-FIR-Filter mit 75 Punkten realisiert. Minimalphase bedeutet, dass das Filter die kürzeste Phasen-Verzögerung unter allen stabilen Filtern mit derselben Amplitudencharakteristik hat, d.h. dieses Filter hat die kürzeste Impulsantwort.

Die Abbildung 4.3 Abbildung zeigt die Signalverarbeitung auf der Beachtron Karte.

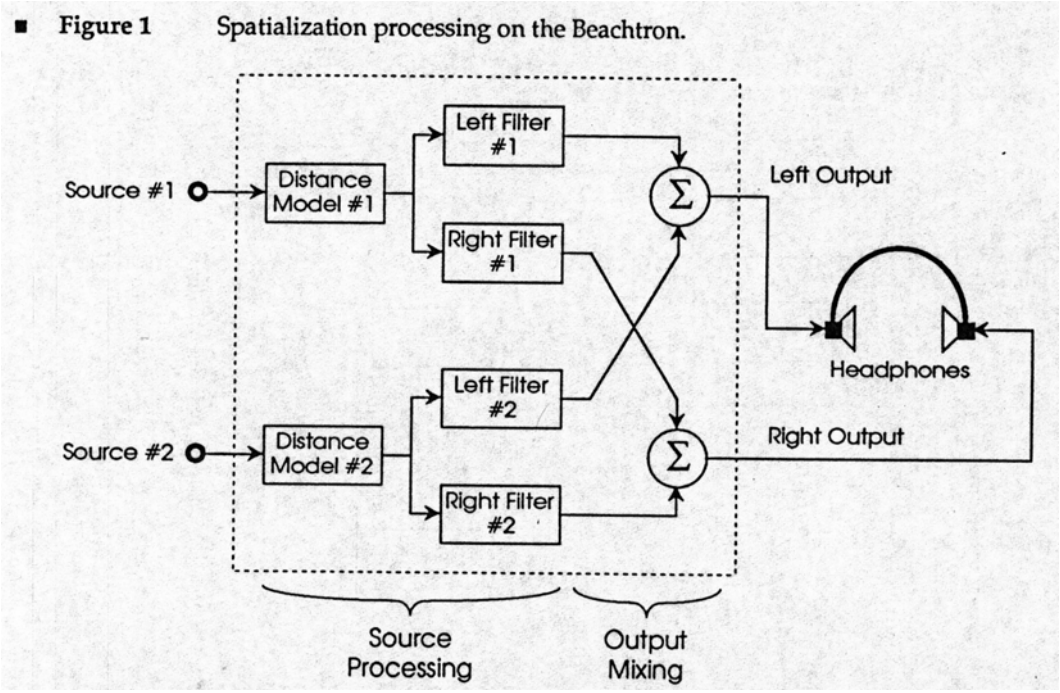


Abbildung 4.3: Signalverarbeitung auf der Beachtron Karte ([Btrn])

Das so genannte „Distance Model“ in der Abbildung 4.3 symbolisiert die Simulation der atmospherischen Verluste. Es ist nichts anderes als ein Dämpfungsglied. Die so genannten „Left Filter“ und „Right Filter“ sind die HRTFs des linken und rechten Ohres.
Die Beachtronkarte ist in der Lage, nicht nur ruhende, sondern auch sich bewegende Schallquellen zu simulieren. Eine Änderung der Position einer Schallquelle erfordert sowohl eine neue HRTF für das linke und rechte Ohr als auch eine entsprechende Reaktion von Seiten des Dämpfungsgliedes. Die Karte kann das Dämpfungsglied und die HRTFs des linken und rechten Ohres bis zu 22 mal in der Sekunde austauschen (alle 46 msec).

Wie ich weiter oben bereits erwähnt habe, benutzt die Beachtron Karte ein drei dimensionales Koordinatensystem. Um den genauen Ort einer Schallquelle anzugeben wird ein 6 dimensionaler Vektor benutzt. Die ersten drei Elemente dieses Vektors geben die Position der Schallquelle an (x,y,z). Die letzten drei Elemente geben die Orientierung der Schallquelle in dem Punkt (x,y,z) an. Es sind drei Elemente für die Orientierung nötig, da diese sich auf alle drei Achsen des Koordinatensystems beziehen. Die Orientierung muss dabei in Gradmaß angegeben werden. Die Abbildung 4.4 zeigt das Koordinatensystem der Beachtron Karte.

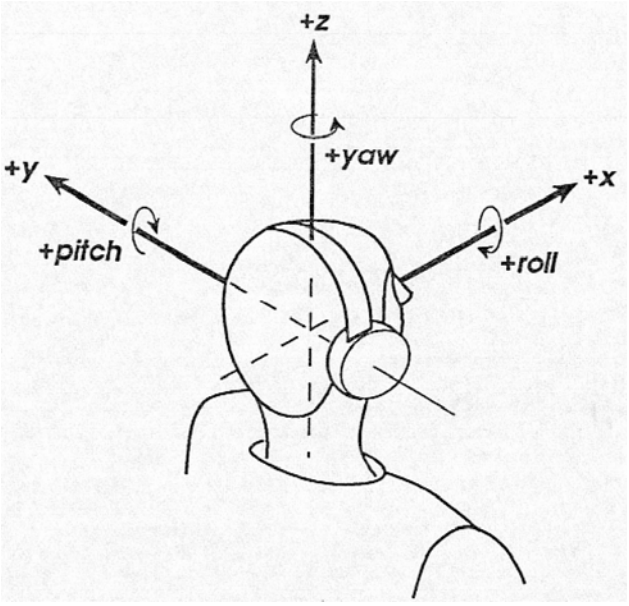


Abbildung 4.4: Koordinatensystem der Beachtron Karte ([Btrn])

Die Beachtron Karte wird mit Hilfe der Programmiersprache C angesteuert. Um die sie steuern zu können, hat der Hersteller dieser Karte fünfzehn C-Funktionen in Form von einer Software Bibliothek mitgeliefert. Ich selber habe für meine beiden Programme nur 12 von 15 Funktionen benötigt. In der Dokumentation der Beachtron Karte befindet sich eine ausführliche Beschreibung sämtlicher Funktionen. Ich habe diese Funktionen auch im Anhang dieser Arbeit abgedruckt.

4.4 Aufgabenstellung

Nun möchte ich die konkrete Aufgabe nennen, die mir mein Betreuer für die Diplomarbeit gestellt hat.

Ich sollte 2 Programme schreiben. Das erste Programm sollte dazu dienen, eine MAA Untersuchung durchzuführen. Das zweite Programm sollte eine „absolute“ Messung durchführen.

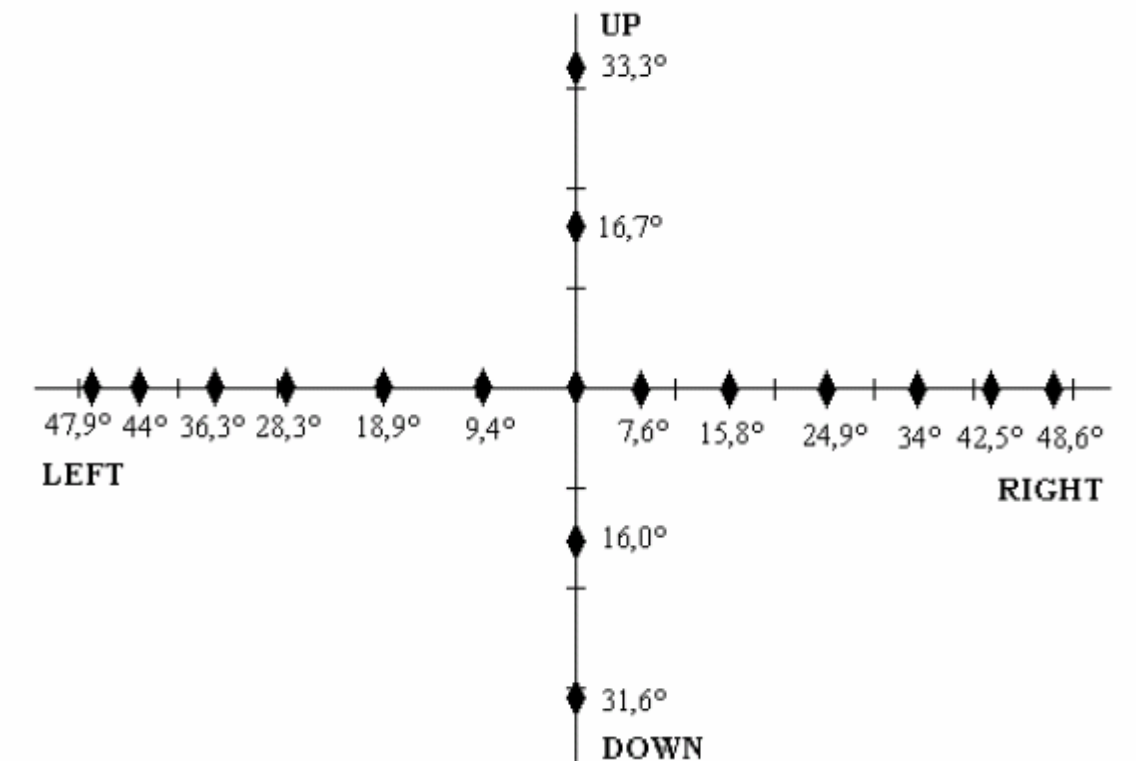
Da mein Betreuer meine Aufgabe schriftlich in Deutsch formuliert hat, möchte ich zunächst seine Aufgabenstellung zitieren. Am Ende werde ich meine Aufgabenstellung ergänzend in eigenen Worten erklären, damit alle Unklarheiten behoben sind.

Der Text von meinem Betreuer ist im Folgenden kursiv gedruckt.

Aufgabe:

Das Programm besteht aus zwei Teilen, kann aber separat geschrieben werden oder auch zusammen. Bei beiden kann man am Anfang den Kopfdurchmesser angeben (falls man das messen möchte), wenn nicht, soll der default Wert benutzt werden. Signal B ist +10 dB, C +6 dB als Signal A.

1.
Simulation von virtuellen Punktschallquellen in der Medianebene und/ oder in der Horizontalebene. Das 2D VAD ist das gleiche (±60 Grad) vorne. Testsignale sind im ersten Schritt auch dieselben: Signal A, B und C (wav). Später können andere Signale verwendet werden, wenn man sie umbenennt. Ausgangspunkt ist das Ergebnis von damals (AVG-Netz von Signal A)



„AVG Netz“, durchschnittliche Auflösung für Signal A, als Ausgangspunkt für das neue Programm.

Am Anfang soll die Frage gestellt werden, ob wir das AVG-Netz benutzen möchten oder eine andere Auflösung simulieren möchten. Während der Simulation ist die Aufgabe der Testperson die Quellen zu identifizieren. Es ist also eine MAA-Untersuchung, denn die Versuchsperson hört am Anfang zwei Quellen wechselweise in 400 ms-Takt. Z.B. eine in dem Punkt 15,8 Grad, die andere in dem Punkt 24,9 Grad (das nennen wir ein Burst-Paar). Wir starten immer im Origo, und man kann wählen in welcher Richtung wir gehen (links, rechts, oben, unten). Die einfache Frage ist, ob die Versuchsperson einen Unterschied wahrnimmt (Antwort Ja) oder nicht (Antwort NEIN). Er darf nur JA sagen, wenn er sicher ist, wenn nicht, sagt er NEIN. Falls er NEIN sagt, bekommt er eine neue Möglichkeit, sodass der nächste Punk simuliert wird, bei diesem Beispiel

34 Grad. Wenn er immer noch NEIN sagt, kommt 42.5 Grad usw. Wenn er JA sagt, dann wird das der neue Referenzpunkt sein… usw. Damit wissen wir am Ende, wie viele Quellen er tatsächlich unterscheiden kann, wir testen damit also das AVG-Netz. Es wird einige geben, die es schaffen, aber auch einige, die es nicht schaffen alle Punkte zu identifizieren. Wir bekommen also Zahlen (%), die aussagen, wie viele es schaffen können mit so vielen Quellen zu arbeiten. Dieses AVG-Netz betrachten wir also als das BEST-CASE, denn 5 vertikal und 13 horizontal wären zu viel. Wir suchen die optimale Auflösung, die von 90% der Personen handelbar ist. Wir werden auch das WORST-CASE finden.
Die Simulation startet immer „stehend", also eine Quelle pulsiert, damit wir testen können, ob sie tatsächlich nichts wahrnehmen, und endet auch damit in dem Referenzpunkt.
Wenn man am Anfang nicht dieses Netz nimmt, dann fragt das Programm, welche Auflösung wir nehmen möchten, z.B. 10 Grad und in welcher Richtung (z.B. rechts). Dann geht das ganze genauso los, aber die Quellen werden in den Punkten 10, 20, 30 usw. simuliert. So können wir schlechtere Auflösungen simulieren (bis wir die optimale Auflösung gefunden haben). Das wäre eigentlich die zweite Phase, denn die Auswertung der Antworten des AVG-Netzes bringt uns dazu, dass wir die optimale Auflösung einschätzen können. Der Mittelpunkt (Origo) ist immer dabei, und das ist immer der erste Referenzpunkt.

2.
Wie gesagt, das oben genannte Programm (oder Programmteil) simuliert Quellen nur in der Horizontalebene und Medianebene. Das Ziel ist es, zu sagen, wie gut ist das AVG-Netz ist (wie viele Menschen könnten es benutzen) und was wäre eine optimale Auflösung für alle (90%?). In einer wirklichen Applikation würden wir dann nicht Punktquellen in diesen Ebenen simulieren, sondern größere Oberflächen nehmen (Fenster). Der Ausgangspunkt für diese Untersuchung ist auch das AVG-Netz und noch dazu die optimale Auflösung. Jetzt ist es aber eine absolute Messung und keine MAA-Untersuchung. Die Schallquelle ist in der Mitte eines Fensters. Es gibt drei Möglichkeiten eine WAVE-Datei abzuspielen: einmal kontinuierlich (0 sec. Pause) oder loop (mit X.sec inzwischen, max 2 sec) oder ohne loop. Das 2D VAD ist immer noch das gleiche.

Die Horizontale und Vertikale Auflösung muss manuell eingegeben werden. BEST CASE ist 7 horizontal, 5 vertikal (Origo inklusive). Mehr kann man nicht angeben. WORST CASE ist 0 vertikal und drei horizontal (Mitte, links und rechts). In diesem Falle wird links und rechts nicht +90° und −90° simuliert, denn das kann jeder (100%) wahrnehmen, sondern auf der 60 Grad Oberfläche simuliert. Die Frage ist wieder: was ist optimal, z.B. 3 vertikal und 5 horizontal.
Die Steuerung kann sein: manuell oder automatisch. Bei manuell ist der Untersuchungsleiter da, und gibt mit der Tastatur an, welches Fenster die Schallquelle ist (mit Ziffern oder Buchstaben). Bei Automatik ist es random. Dann muss der Leiter eine bestimme Taste nach der Antwort der Versuchsperson drücken um in dem Programm fortzufahren. Außerdem notiert er, welches Fenster aktiv war und wie die Antwort lautete. Bei der Antwort müssen die Versuchspersonen angeben, aus welchem Fenster das Geräusch kam. Die Antwort kann entweder richtig sein, falsch sein oder ein benachbartes Fenster sein.

Nun möchte ich ergänzend in eigenen Worten meine Aufgabenstellung erklären, damit alle Unklarheiten beseitigt sind.

Das erste Programm, das ich schreiben sollte, sollte einer MAA Untersuchung dienen. Es sollte dem Benutzer die Möglichkeit bieten, zwischen einer eigenen, selbst festgelegten Auflösung oder einer default Auflösung zu wählen. Mit default Auflösung ist hier das AVG-Netz gemeint.

Die Abkürzung AVG steht für „average“. Das AVG-Netz ist aus einer früheren MAA Untersuchung hervorgegangen, die im Rahmen des GUIB-Projektes durchgeführt wurde. Damals hatten 50 Testpersonen (25 Weibliche und 25 Männliche) an der Untersuchung teilgenommen. Es wurden 3 verschiede Untersuchungen mit jeweils 3 verschiedenen Testsignalen durchgeführt.

Das erste Testsignal (Signal A) war digital erzeugtes breitbandiges weißes Rauschen. Das zweite Testsignal entstand durch passive Tiefpaß-Filterung von Signal A mit einer oberen Grenzfrequenz von 1500 Hz. Das dritte Signal entstand durch Hochpaß-Filterung von Signal A mit einer unteren Grenzfrequenz von 7000 Hz. Um dafür zu sorgen, dass alle drei Signale subjektiv die gleiche Lautstärke besaßen, wurde Signal B um +10 dB und Signal C um +6 dB gegenüber Signal A verstärkt.

Die Abbildung 4.5 zeigt den Frequenzgang der Testsignale:

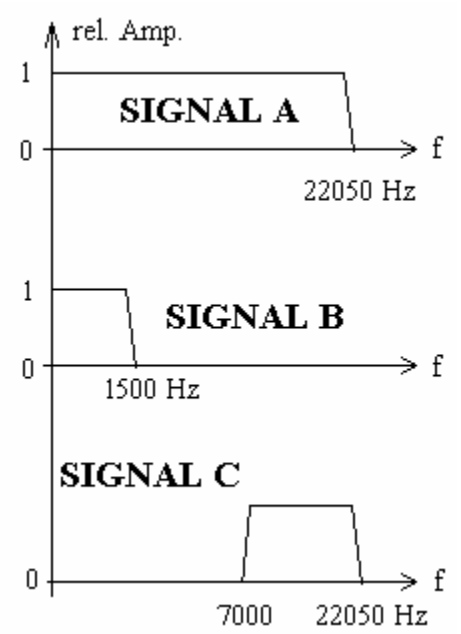


Abbildung 4.5: Frequenzgang der Signale. Signal A ist weißes Rauschen mit 16 bit und 44.1 kHz Auflösung. Signal B und C sind werden Tiefpaß bzw. Hochpaßfilterung mit einer Grenzfrequenz von 1500 und 7000 Hz erzeugt. ([Wers-diss])

Das AVG-Netz ist der gerade wahrnehmbare Unterschied für Signal A. Es repräsentiert das Durchschnittsergebnis der 50 damals teilnehmenden Personen an der MAA Untersuchung.

Mein erstes Programm sollte also dem Nutzer die Möglichkeit geben, entweder das AVG-Netz mit neuen Versuchspersonen und den gleichen Testsignalen (Signal A, B, C) nochmals zu testen oder in selbst festgelegten Gradschritten eine zweite MAA Untersuchung durchzuführen.

Dabei sollte es dem Untersuchungsleiter ermöglichen, die Diskriminationsfähigkeit in vier Schritten (nach rechts, links, oben und unten) in der Horizontalebene bzw. in der Medianebene zu untersuchen. Die Rauschimpulspaare sollten in einem Zwischenzeitraum von 400 ms erfolgen. Der erste Impuls sollte immer von der Referenzrichtung (Anfangs vom Ursprung) kommen, die anderen von einer seitlichen Auslenkung oder von einem bestimmten Erhebungswinkel.

Die Abbildung 4.6 verdeutlicht die Signaldarbietung während der MAA Untersuchung.

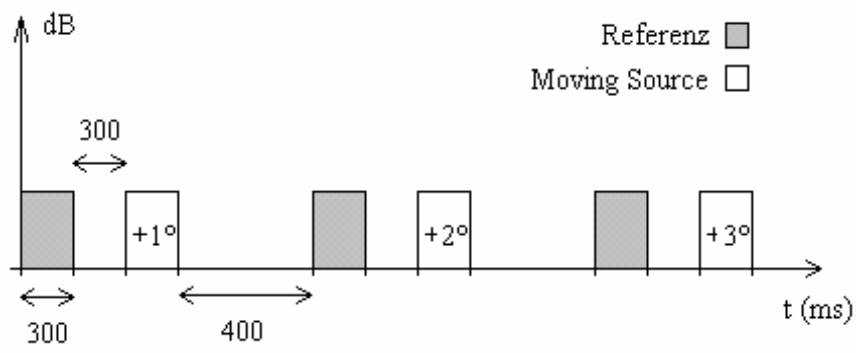


Abbildung 4.6: Signaldarbietung während der MAA-Untersuchung. Die 300 ms Rauschbursts-Paare erfolgen in 400 ms takt. Zwischen den Bursts zum MAA-Vergleich sind 300 ms. Der erste Impuls kommt immer von einer Referenzrichtung, der andere von einer seitlichen oder vertikalen Auslenkung. ([Wers-diss])

Wie man an der obigen Abbildung sieht, erfolgen die 300 ms Rauschbursts-Paare in 400 ms Takt. Zwischen den Bursts zum MAA-Vergleich sind 300 ms. Der erste Impuls kommt immer von einer Referenzrichtung, der andere von einer seitlichen oder vertikalen Auslenkung (in diesem Beispiel in +1° Schritte). Sobald die Versuchsperson einen Unterschied zwischen den Bursts feststellt, wird die Referenzquelle zu dem Punkt verschoben, an dem die sich bewegende Schallquelle zum Zeitpunkt der Wahrnahme des Unterschieds befand. Bei jedem Wechsel des Referenzpunktes (auch ganz zu Beginn) befinden sich beim ersten Rauschburst-Paar beide Quellen am selben Ort. Damit will man testen, ob die Versuchsperson tatsächlich einen Unterschied wahrnimmt oder nur denkt, einen Unterschied wahrgenommen zu haben.

Ich möchte hier noch etwas zu den drei Testsignalen sagen. Man hat sich bei der Untersuchung aus folgenden Gründen für weißes Rauschen und deren gefilterte Version als Testsignal entschieden: Weißes Rauschen ist breitbandig, hat einen flachen Frequenzgang, unbegrenzte Energie und kann beliebig lange andauern - im Gegensatz zu Dirac-Impulsen. Deswegen sind solche Signale und deren gefilterte Versionen gut für die Untersuchungen geeignet. Durch lineare Filterung lassen sich aus weißem Rauschen regellose Vorgänge mit beliebigen gewünschten Bandbreiten und Spektren herstellen, wie auch in unserem Fall.

Ich habe sowohl das erste, als auch das zweite Programm so programmiert, dass die Untersuchungsergebnisse jeder Versuchsperson am Ende automatisch in eine Textdatei abgelegt wurden. Dabei musste man im Programm nur den Namen der Versuchsperson eingeben, um die einzelnen Textdateien voneinander zu unterscheiden. So konnte man am Ende einer Untersuchungsreihe mit Hilfe der Textdateien eine Auswertung der Ergebnisse durchführen.

Um eine Untersuchung mit dem ersten Programm durchzuführen, mussten zu Beginn folgende Parameter eingestellt werden:

- Angabe der Abstände beider Ohren oder Benutzung des default Wertes (um die interauralen Laufzeitunterschiede der Kopfgröße anzupassen)
- Wahl der Auflösung in Gradmaß oder Benutzung des AVG-Netzes
- Wahl des Testsignals (Entweder Signal A, B, C oder eine beliebige andere Wave-Datei)
- Verstärkung des Testsignals A oder Benutzung des default Wertes für Signal A (0 dB) (optional; dabei war Signal B immer um +10 dB und Signal C immer um +6 dB stärker als Signal A)
- Eingabe des Namens der Versuchsperson (um die Textdatei mit den Untersuchungsergebnissen anzulegen)

Wie oben aufgelistet ist, kann man mit einem speziellen Befehl, der in der 16 Bit Beachtron Bibliothek definiert war, den Abstand beider Ohren angeben. Damit bietet der Hersteller dem Nutzer die Möglichkeit, dir HRTFs individueller zu machen. Die Abbildung 4.7 verdeutlicht dies.

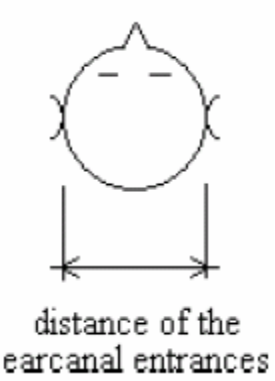


Abbildung 4.7: Distanz zwischen den Gehöreingängen der Versuchsperson. ([Wers-diss])

Hatte man sämtliche oben genannte Parameter eingestellt, konnte man mit der Untersuchung beginnen.

Mein zweites Programm sollte dem Nutzer die Möglichkeit geben, eine „absolute“ Messung durchzuführen. Dazu sollte es das 2D VAD in gleich große Fenster aufzuteilen. Bei der kleinsten Auflösung sollte das 2D VAD in 3 Horizontale und einem Vertikalen Fenster aufgeteilt werden (3x1). Bei der größten Auflösung sollte es in 7 Horizontale und 5 Vertikale Fenster aufgeteilt

werden (7x5). Insgesamt gab es also somit 25 verschiedene Möglichkeiten das 2D VAD aufzuteilen.

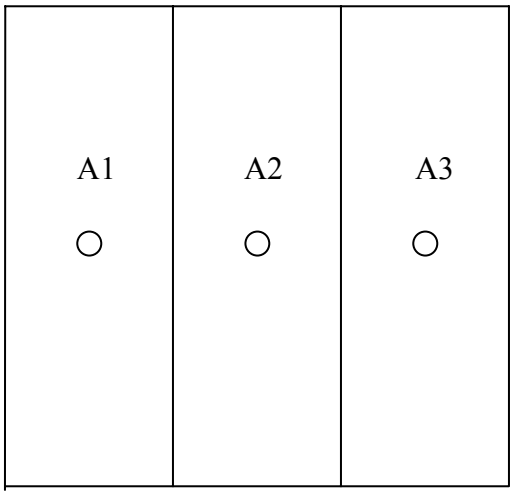


Abbildung 4.8: Auflösung: 3x1.

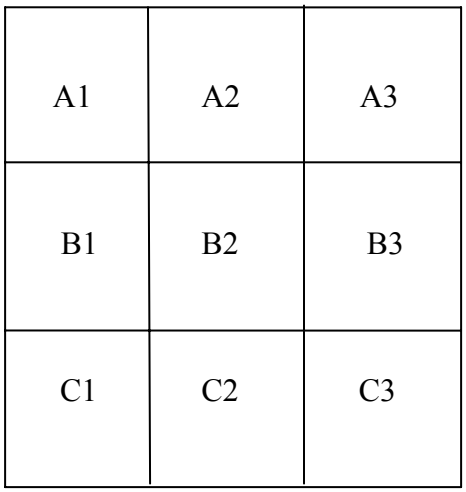


Abbildung 4.9: Auflösung: 3x3.

An der Abbildung 4.8 und 4.9 sieht man 2 Beispiele für eine Aufteilung des 2D VADs. Die Schallquelle sollte sich dabei immer in der Mitte des Fensters befinden. Die Versuchsperson wusste, welche Auflösung das 2D VAD haben wird und musste während der Untersuchung nur die Koordinaten des Fensters nennen (A1, B2, C3 etc.). Dabei wurden ihr die gleichen Testsignale (Signal A, B, C) vorgespielt, wie während der MAA Untersuchung auch. Der Versuchsleiter konnte zwischen folgender Art und Weise der Signaldarbietung wählen:

- kontinuierliche Wiedergabe des Testsignals (Die Versuchsperson hat das Testsignal solange gehört, bis eine Antwort von ihr kam.)
- 300 ms lange Wiedergabe des Signals (ohne Wiederholung)
- 300 ms lange Wiedergabe des Signals (mit Wiederholung; bis zu 20 Wiederholungen möglich). Dabei konnten die Pause zwischen den einzelnen Wiederholungen vom Versuchsleiter selbst gewählt werden (1 ms – 7000 ms)
- manuelle Auswahl des Fensters oder automatische Auswahl des Fensters (mittels Zufallsgenerators)

Ich habe den Zufallsgenerator so programmiert, dass pro Runde ein bestimmtes Fenster nur ein einziges Mal dran kam. Erst wenn alle anderen Fenster durchlaufen waren, wurden sämtliche Fenster noch mal gemischt.

Außer den oben genannten Parametern konnte man, wie im ersten Programm auch folgende Parameter einstellen:

- Angabe der Abstände beider Ohren oder Benutzung des default Wertes (um die interauralen Laufzeitunterschiede der Kopfgröße anzupassen)
- Wahl des Testsignals (Entweder Signal A, B, C oder eine beliebige andere Wave-Datei)
- Verstärkung des Testsignals A oder Benutzung des default Wertes für Signal A (0 dB) (optional; dabei war Signal B immer um +10 dB und Signal C immer um +6 dB stärker als Signal A)
- Eingabe des Namens der Versuchsperson (um die Textdatei mit den Untersuchungsergebnissen anzulegen)

Die Textdatei, die für jede Versuchsperson vom Programm automatisch angelegt wurde, enthielt Informationen über die eingestellten Parameter und eine Matrix, aus der hervorging, welches Fenster aktiv war und wie die Antwort der Versuchsperson lautete.

Ich habe etwa 8 Wochen gebraucht, um beide Programme zu schreiben, zu testen und sämtliche Fehler zu beheben. Als ich fertig war, beauftragte mich mein Betreuer mit Hilfe der beiden Programmen ausführliche Untersuchungen durchzuführen.

Dazu hat er alle Studenten aus seiner eigenen Vorlesung eingeladen um an der Untersuchung teilzunehmen. Im folgenden Abschnitt möchte ich nun über diese Untersuchungen berichten und deren Auswertung und Ergebnisse darstellen. Anschließend werde ich die Ergebnisse für sämtliche Untersuchungen diskutieren.

4.5 Unsere Untersuchung

Wir haben insgesamt drei Untersuchungen durchgeführt. Die erste Untersuchung war eine MAA Untersuchung, in der wir das AVG-Netz getestet haben. An dieser Untersuchung nahmen insgesamt 42 Studenten teil (40 Männliche und 2 Weibliche).

Die nächsten beiden Untersuchungen waren absolute Untersuchungen. Dabei haben wir in der zweiten Untersuchung das 2D VAD in 3 vertikale und 3 Horizontale Fenster aufgeteilt und in der dritten Untersuchung in 2 vertikale und 5 horizontale. An der zweiten Untersuchung nahmen insgesamt 29 Studenten teil (28 Männliche und 1 Weibliche) und an der dritten nahmen 28 Studenten teil (27 Männliche und 1 Weibliche).

Über 95% der Versuchsteilnehmer waren zwischen 20 und 25 Jahre alt. Das Alter spielt bei einer solchen Untersuchung aber keine so überragende Rolle, da frühere Untersuchungen gezeigt haben, dass die Lokalisationsfähigkeit des Menschen zwischen dem 18 und 45 Lebensjahr keine großen Unterschiede aufweist.

Frühere Untersuchungen, die im Rahmen des GUIB Projektes durchgeführt wurden, haben auch gezeigt, dass blinde Personen bei einer Simulation (unter Kopfhörerbedingungen) nicht unbedingt besser lokalisieren können, als sehende. Daher ist es keine Verfälschung der Ergebnisse, wenn an unserer Untersuchung nur Personen mit gesunder Sehfähigkeit teilgenommen haben.

4.5.1 MAA Untersuchung

Wie bereits oben erwähnt, haben wir in der MAA Untersuchung das AVG Netz getestet. Dabei wurden pro Person jeweils drei Testreihen durchgeführt und zwar mit allen drei Testsignalen

Im Folgenden werde ich jetzt mit Hilfe von Tabellen und Diagrammen eine Auswertung der Ergebnisse durchführen.

Wie ich bereits in dem letzten Abschnitt erläutert hatte, befinden sich bei jedem Wechsel des Referenzpunktes (auch ganz zu Beginn) beim ersten Rauschburst-Paar beide Quellen am selben Ort. Damit will man testen, ob die Versuchsperson tatsächlich einen Unterschied wahrnimmt oder nur denkt, einen Unterschied wahrgenommen zu haben. In der folgenden Auswertung werde ich zu jedem Punkt innerhalb des AVG-Netzes eine Fehlerangabe machen. Diese Fehlerangabe bedeutet, das die Versuchsperson beim Wechsel des Referenzpunktes zu dem betreffenden Punkt beim ersten Rauschburst-Paar (wo sich beide Quellen am selben Ort befanden) einen Unterschied in der Lokalisation wahrnahm, wo in Wirklichkeit keiner war.

Das AVG-Netz hat auf der Linken und Rechten Seite jeweils 6 Punkte und oben und unten jeweils 2 Punkte. Ich werde in den folgenden Diagrammen die jeweiligen Punkte mit *„1. Punkt Rechts, 2. Punkt Oben“* etc. bezeichnen.

Die nun folgenden 4 Diagramme zeigen, wie häufig die einzelnen Punkte auf den vier Achsenabschnitten von den 42 Versuchsteilnehmern erkannt wurden. Dabei kann ein einzelner Punkt pro Signal maximal 42 Mal erkannt worden sein, da es nur 42 Versuchsteilnehmer gab.

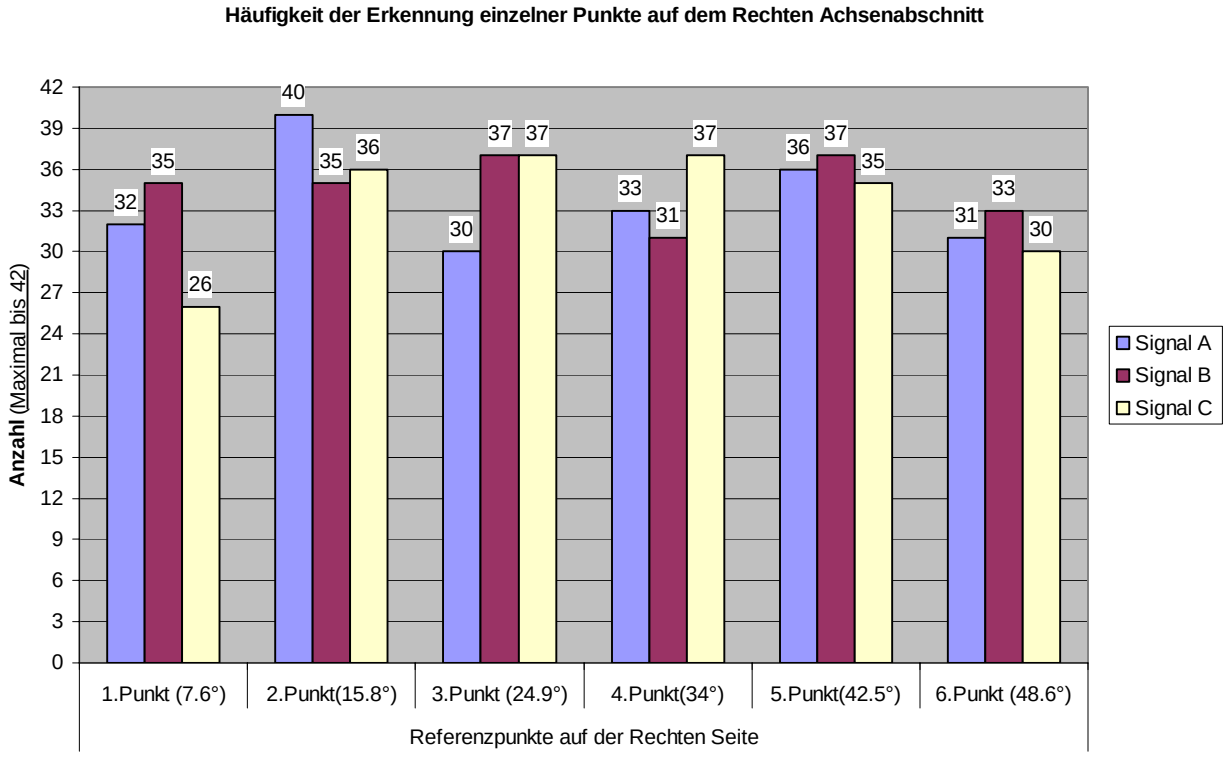


Abbildung 4.10

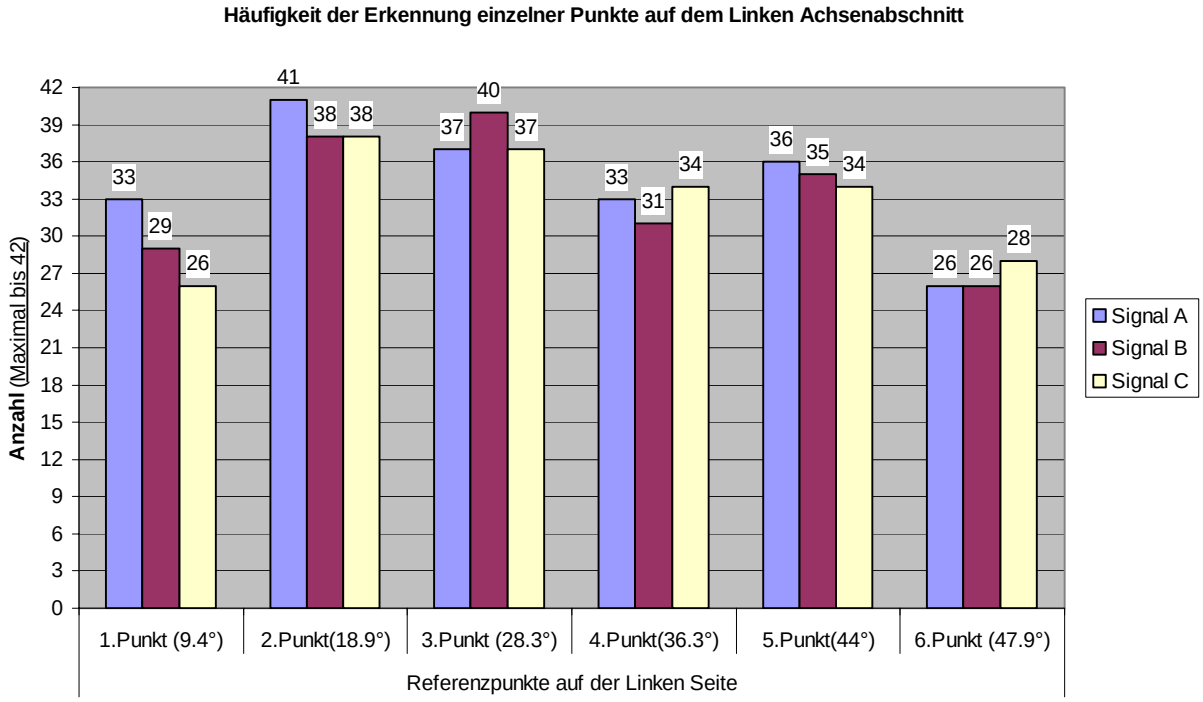


Abbildung 4.11

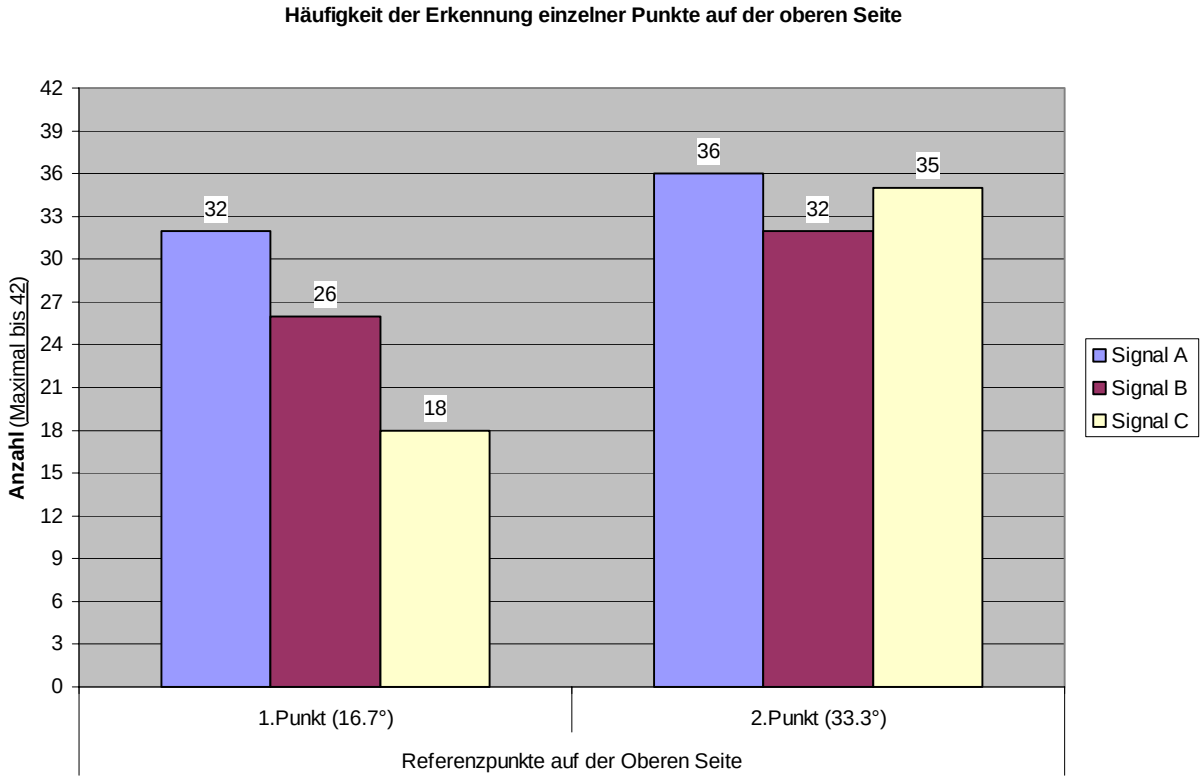


Abbildung 4.12

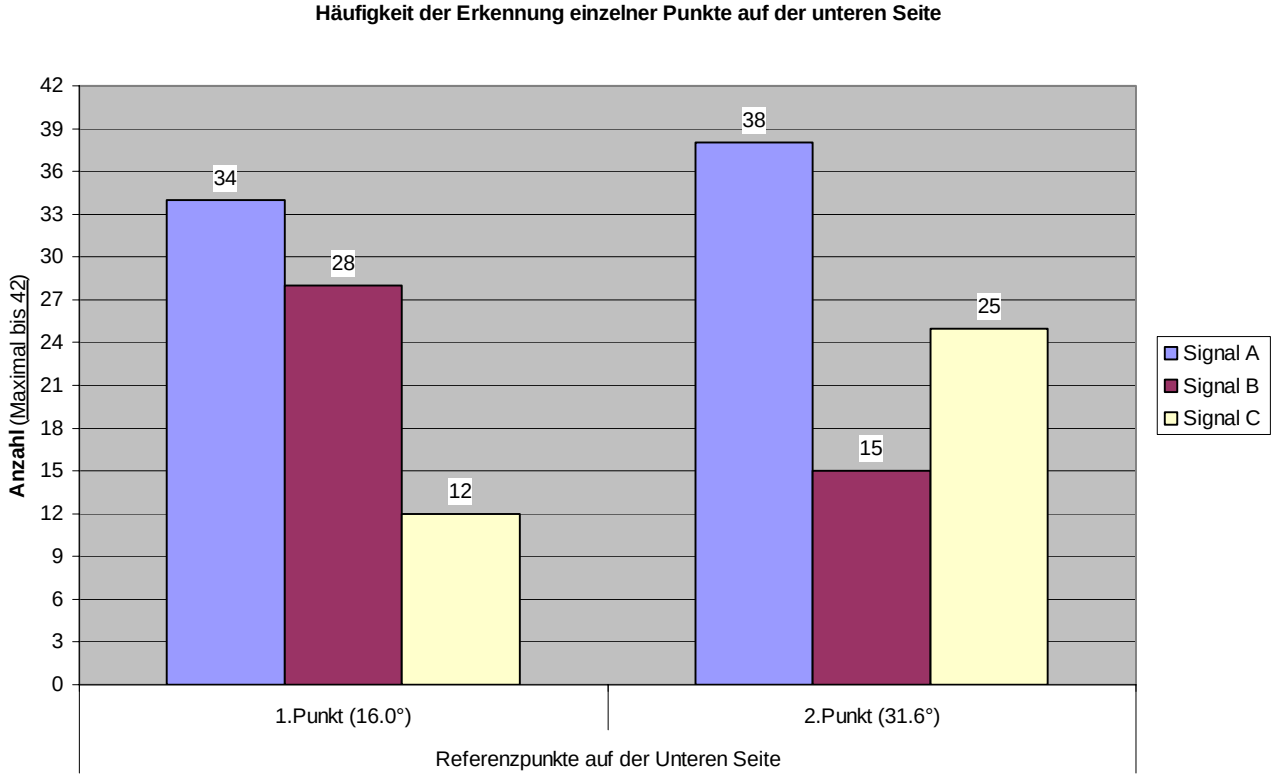


Abbildung 4.13

In den obigen 4 Diagrammen wurde jeder Punkt des AVG-Netzes einzeln betrachtet und es wurde aufgezählt, wie oft jeweils ein bestimmter Punkt von den Versuchsteilnehmern erkannt wurde.
In den nun folgenden Diagrammen möchte ich darstellen, welche Anzahl von Punkten jeweils wie viele Versuchsteilnehmer erkannt haben. D.h. es wird dargestellt, wie viele Leute alle 16 Punkte des AVG-Netzes erkannt haben, wie viele nur 15 erkannt haben usw. Dabei werde ich genauer differenzieren, in dem ich die 6 linken und 6 rechten Punkte, die allesamt in der Horizontalebene liegen, zusammenfasse und angebe, wie viele Versuchsteilnehmer alle 12 Punkte in der Horizontalebene erkannt haben, wie viele nur 11 erkannt haben usw.
Das gleiche werde ich auch mit der Medianebene machen, in dem ich die 2 oben und 2 unten liegenden Punkte zusammenfasse und angebe, wie viele Versuchsteilnehmer alle 4 Punkte erkannt haben und wie viele weniger.

Ich möchte hier anmerken, dass bei dieser Darstellungsweise nicht gesagt werden kann, welche Punkte nun genau erkannt wurden. Wenn zum Beispiel in der Horizontalebene insgesamt 5 Versuchsteilnehmer 9 Punkte erkannt haben, können bei jedem Versuchsteilnehmer die Punkte unterschiedlich in der Horizontalebene verteilt sein. Diese Darstellungsweise sagt also nichts über die Verteilung der Punkte aus, sondern macht lediglich eine Aussage darüber, welche Punktzahl von wie vielen Versuchsteilnehmern erkannt wurden.

Im Folgenden werde ich jetzt die Ergebnisse zunächst für die Horizontalebene, dann für die Medianebene und zuletzt für das gesamte AVG-Netz zusammenfassend darstellen.

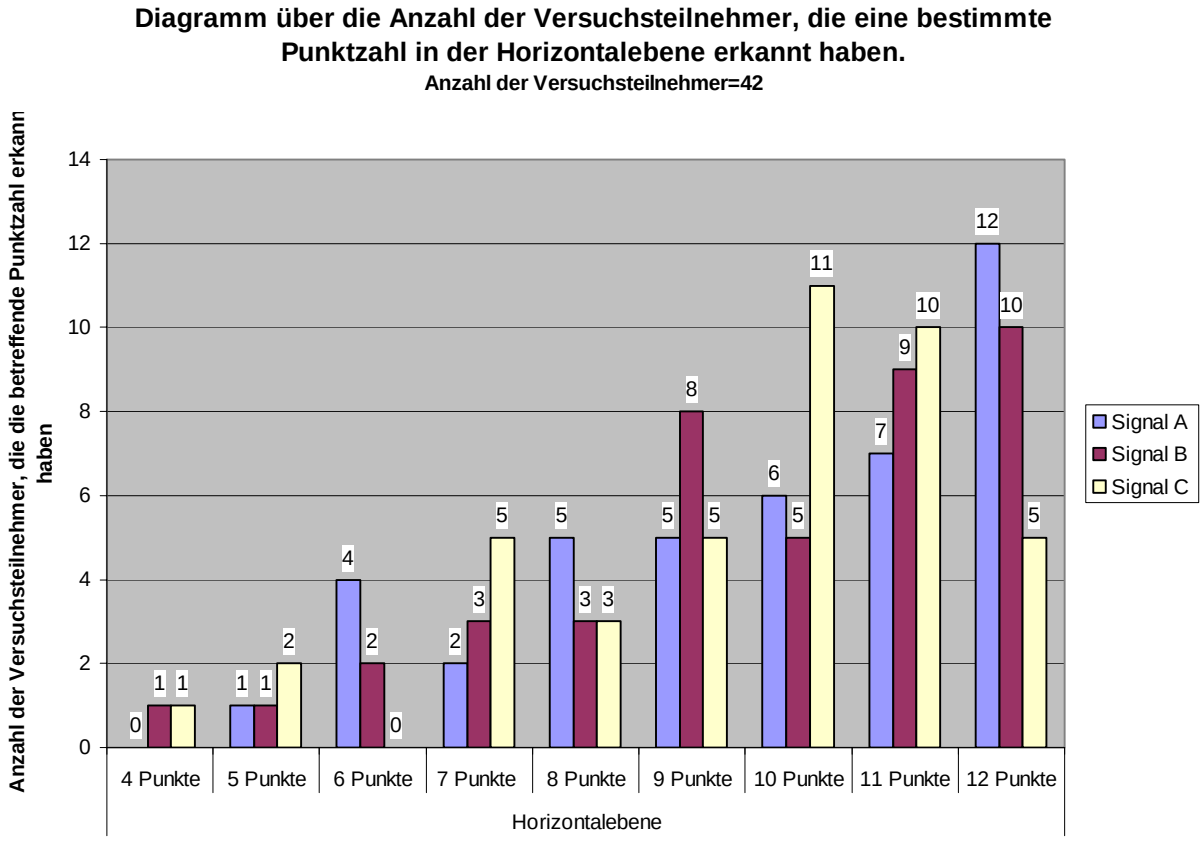


Abbildung 4.14

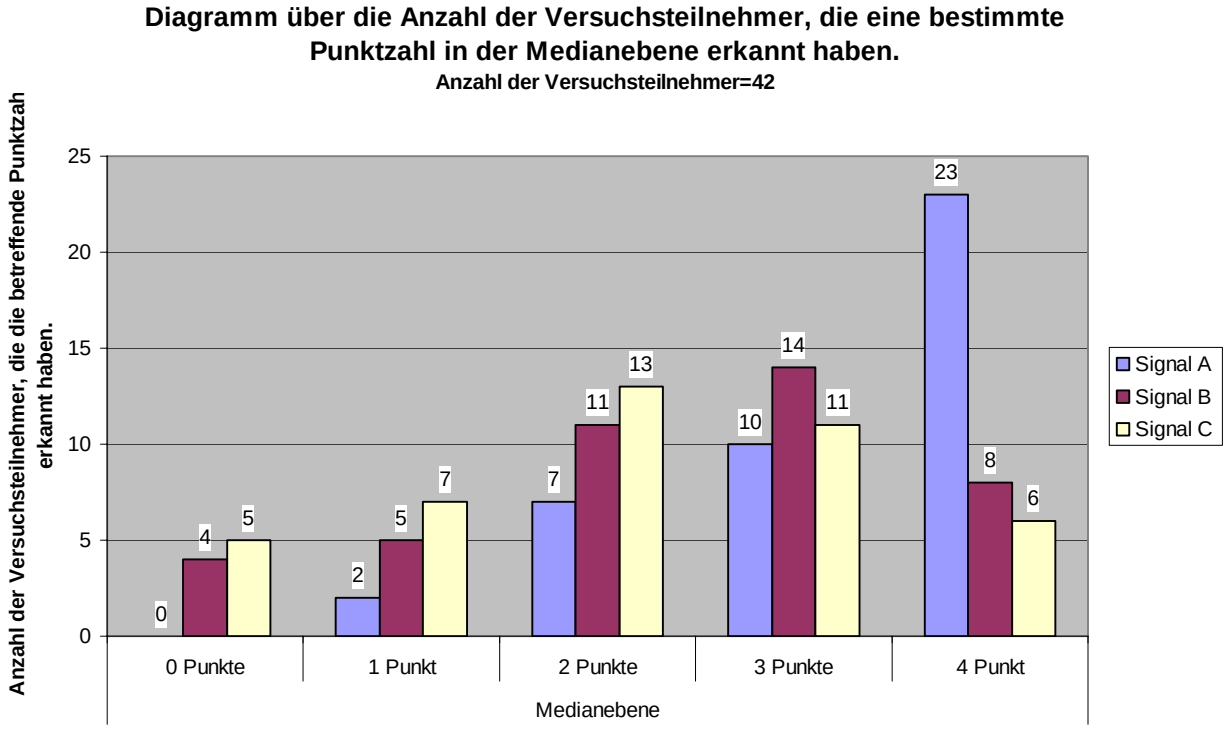


Abbildung 4.15

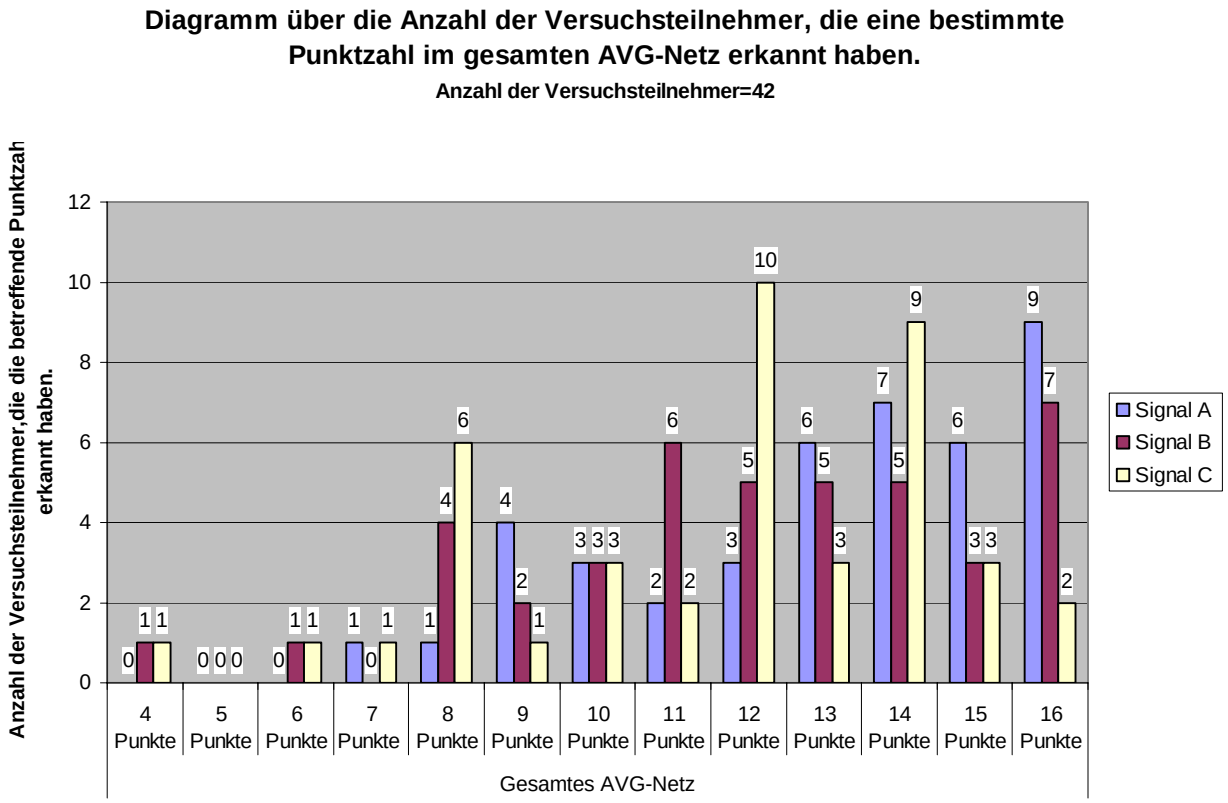


Abbildung 4.16

Ich habe während der MAA Untersuchung 30 der 42 teilnehmenden Versuchspersonen auch gefragt, ob diese mir sagen können, in welcher Richtung sich die Schallquellen während der Untersuchung bewegt haben. Dabei habe ich pro Signal für alle 4 Richtungen notiert, ob die Versuchsperson die Richtung erkannt hat oder nicht. Im nächsten Diagramm möchte ich die Ergebnisse dieser Befragung darstellen.

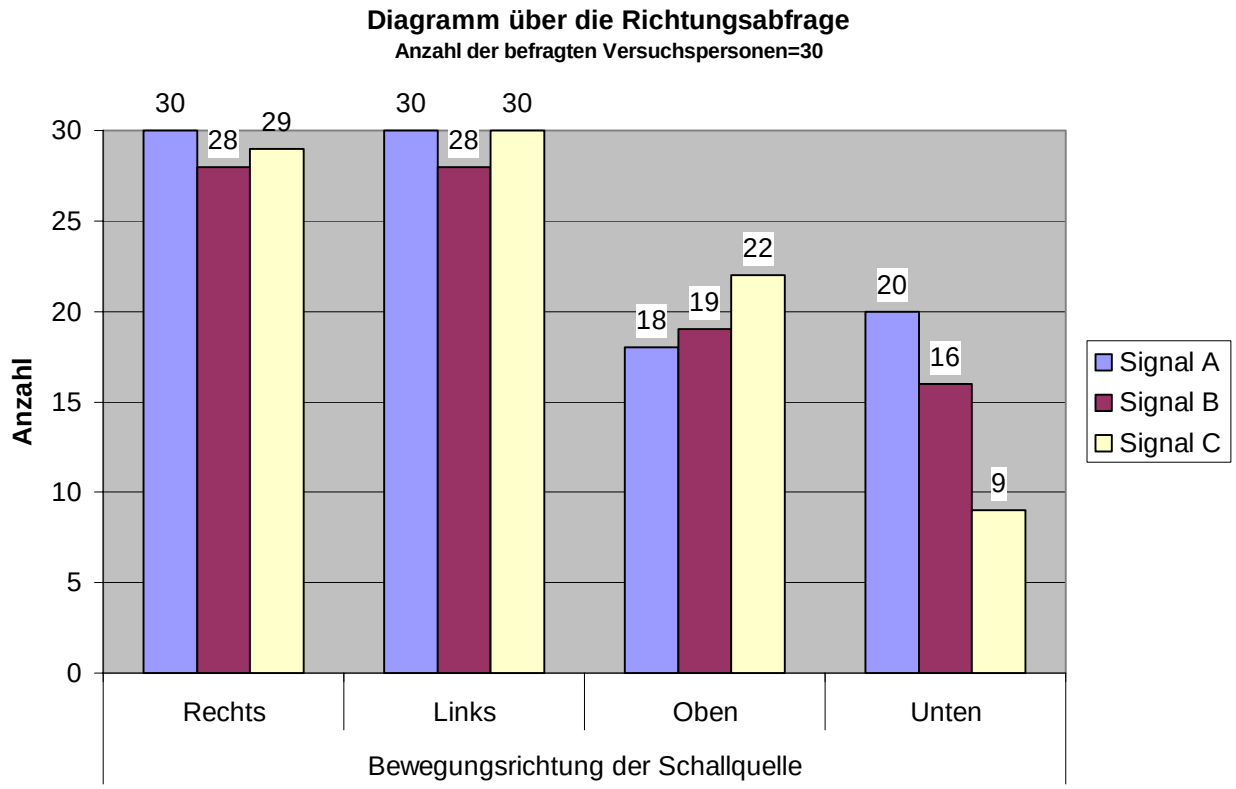


Abbildung 4.17

Wie man an der Abbildung 4.17 sieht, sind die Ergebnisse innerhalb der Horizontalebene besonders gut ausgefallen. Für Signal A haben sogar alle befragten Versuchsteilnehmer für Links und Rechts die richtige Antwort gegeben. Deutlich schlechter sind dagegen die Ergebnisse in der Medianebene. Dort haben im besten Falle etwa 1/3 der befragten Versuchsteilnehmer nicht erkannt, in welcher Richtung sich die Schallquellen bewegen.
Eine genauere Diskussion der bisherigen Ergebnisse wird später erfolgen.

Abschließend für die MAA-Untersuchung möchte ich nun für jeden Punkt innerhalb des AVG-Netzes die Fehler darstellen. Die Abbildung 4.18 und 4.19 macht zu jedem Punkt innerhalb des AVG-Netzes eine Fehlerangabe. Diese Fehlerangabe bedeutet, das die Versuchsperson beim Wechsel des Referenzpunktes zu dem betreffenden Punkt beim ersten Rauschburst-Paar (wo sich beide Quellen am selben Ort befanden) einen Unterschied in der Lokalisation wahrnahm, wo in Wirklichkeit keiner war.

Die beiden folgenden Diagramme zeigen nun diese Fehler innerhalb der Horizontal und Medianebene.

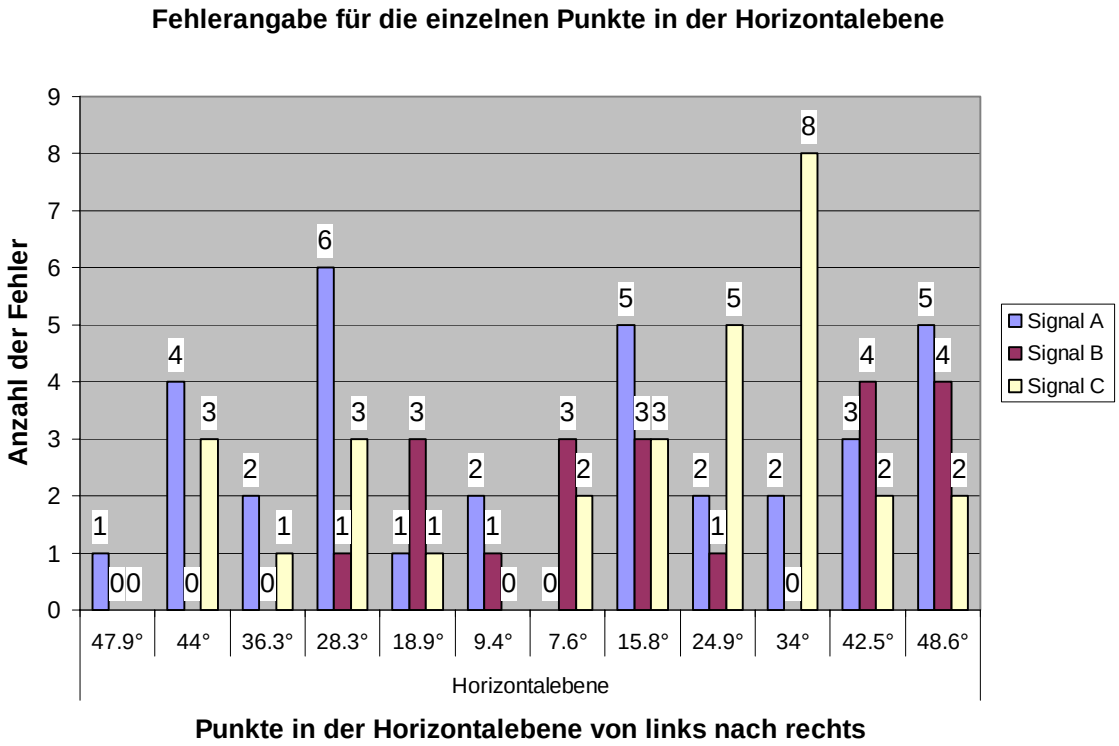


Abbildung 4.18

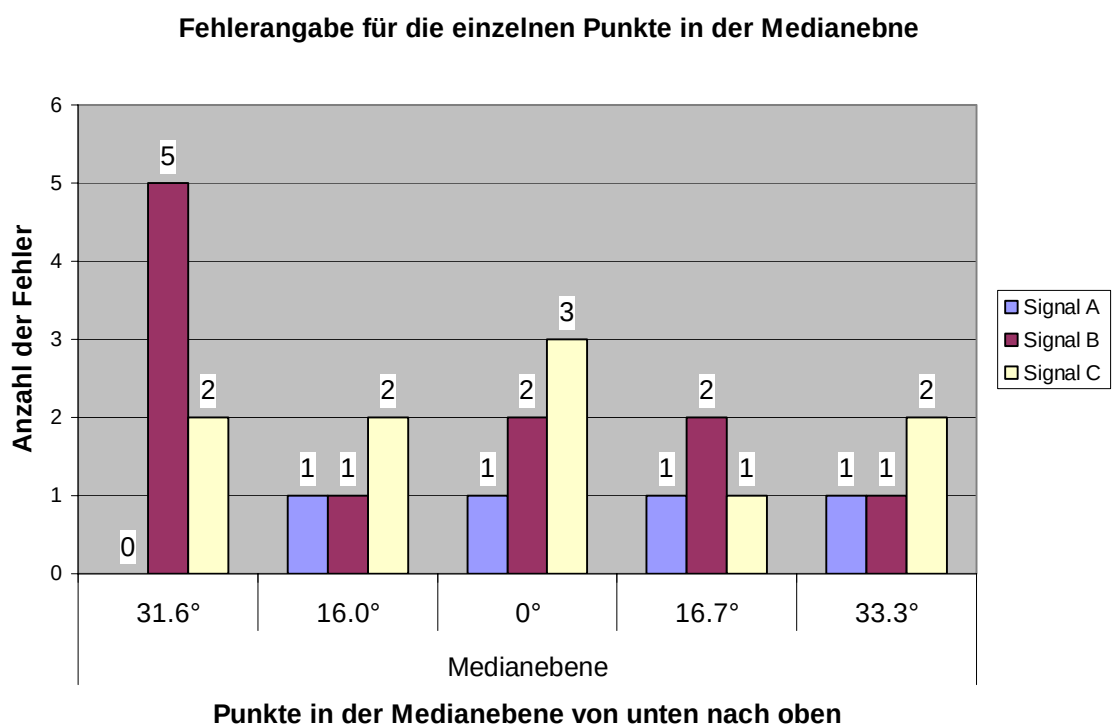


Abbildung 4.19

4.5.2 Absolute Messung

Nun werde ich mit Hilfe von Diagrammen die Untersuchungsergebnisse der absoluten Messung darstellen. Bei der absoluten Messung haben wir zwei verschiede Versuchsreihen durchgeführt.

Bei der ersten Versuchsreihe war das 2D VAD in 3 Horizontale und 3 Vertikale Fenster aufgeteilt (Auflösung: 3x3). Bei der zweiten Versuchsreihe war das 2D VAD in 5 Horizontale und 2 Vertikale Fenster aufgeteilt.

An der ersten Versuchsreihe nahmen 29 Personen teil und an der zweiten Versuchsreihe nahmen 28 Personen teil.

Ich habe jeweils für beide Versuchsreihen mit jeder Versuchperson dieselbe Untersuchung zweimal durchgeführt. Somit hat jede Versuchsperson pro Versuchsreihe für jedes Fenster 2 Antworten gegeben. Sämtliche Antworten einer Versuchsperson wurden am Ende in eine Textdatei abgelegt, aus der hervorging, welches Fenster aktiv war und wie die Antwort der Versuchsperson lautete.

Ich habe für die Auswertung die Antworten der Versuchspersonen folgendermaßen klassifiziert. Entweder war die Antwort richtig, falsch oder es wurde ein benachbartes Fenster genannt. Mit benachbartem Fenster meine ich, dass entweder ein direkt drüber oder direkt drunter liegendes Fenster genannt wurde. Wenn die Versuchsperson ein benachbartes Fenster genannt hat, hat sie nur die horizontale Richtung des Schallereignisses richtig erkannt, nicht aber die vertikale Richtung. Bei einer falschen Antwort hat sie entweder nur die vertikale Richtung richtig erkannt oder weder die vertikale noch die horizontale Richtung richtig erkannt.

Ich will das anhand eines Beispiels verdeutlichen. Die Abbildung 4.21 zeigt das 2D VAD mit einer 3x3 Auflösung.

A1	A2	A3
B1	B2	B3
C1	C2	C3

Abbildung 4.20: 2D VAD; Auflösung 3x3;

Wenn nun das Fenster B2 aktiv wäre und die Antwort der Versuchsperson A2 lauten würde, wäre diese Antwort ein benachbartes Fenster. Auch bei C2 als Antwort wäre das Fenster benachbart. Würde die Versuchsperson aber B1 oder A3 antworten, wären diese Antworten falsch.

Ich habe für beide Auflösungen die Namen der Fenster folgendermaßen vergeben.
Die Buchstaben geben die vertikale Position eines Fensters an, dabei bekommt die oberste vertikale Position den Buchstaben „A“, die weiter drunter liegenden Positionen die Buchstaben „B“ und „C“ usw.
Die Zahlen geben die Horizontale Position eines Fensters an. Dabei bekommt die ganz links gelegene horizontale Position die Zahl „1“ zugewiesen, die weiter rechts liegenden Positionen dagegen höhere Zahlen zugewiesen.

Nun möchte ich zunächst die Versuchsergebnisse für die Auflösung 3x3 darstellen. Da mit jeder Versuchsperson dieselbe Untersuchung zweimal durchgeführt wurde, können pro Fenster maximal 2x29=58 richtige, benachbarte oder falsche Antworten gegeben worden sein. Dieser Extremfall kam natürlich bei keinem Fenster vor, sondern die Antworten waren auf richtige, falsche und benachbarte Antworten verteilt. Die Summe aller drei Antworten pro Fenster ergibt bei der Auflösung 3x3 aber stets 58.
Bei der Auflösung 5x2 nahmen nur 28 Personen an der Untersuchung teil, deshalb ist dort die Summe aus richtigen, benachbarten und falschen Antworten 2x28=56.

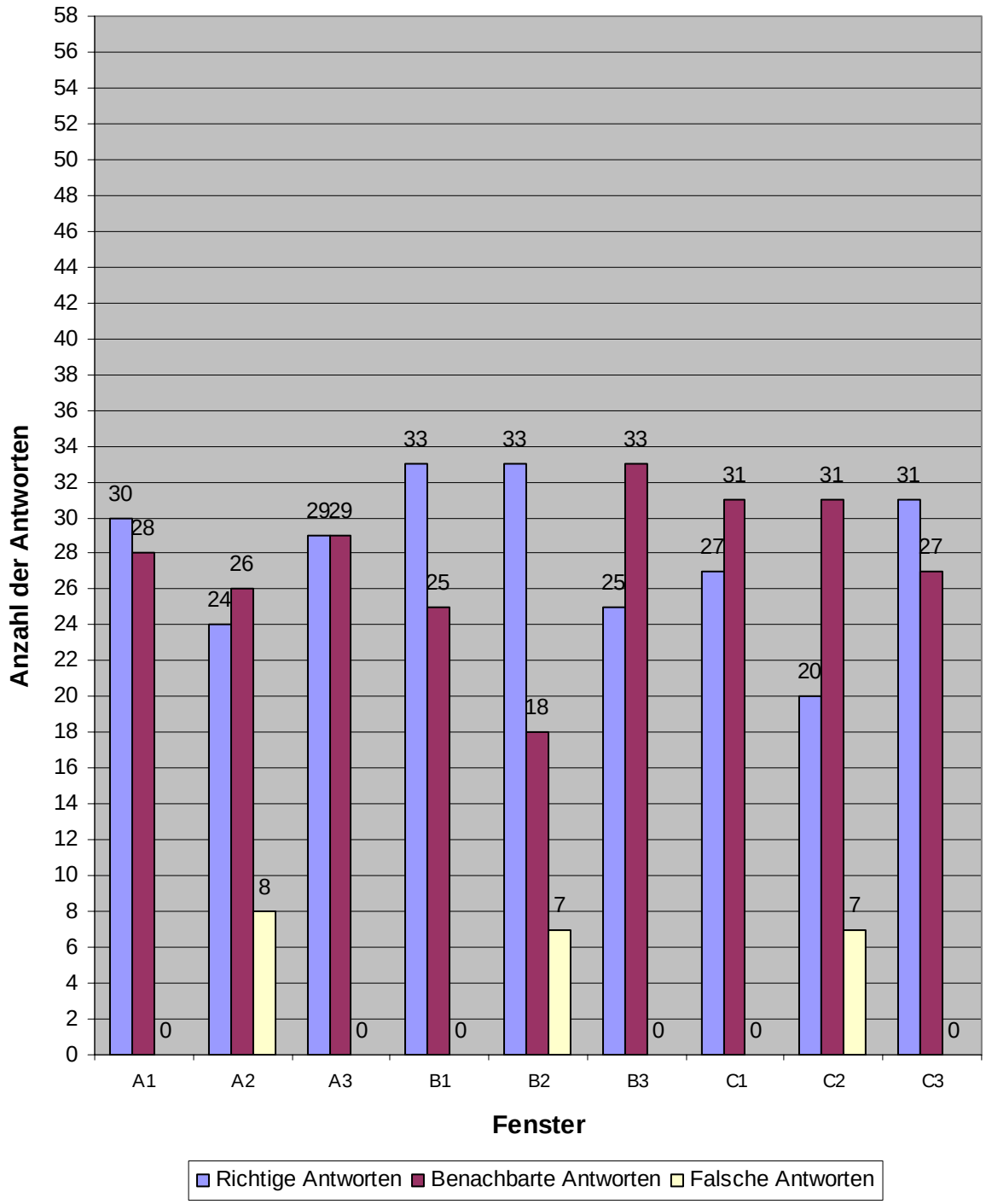


Abbildung 4.21

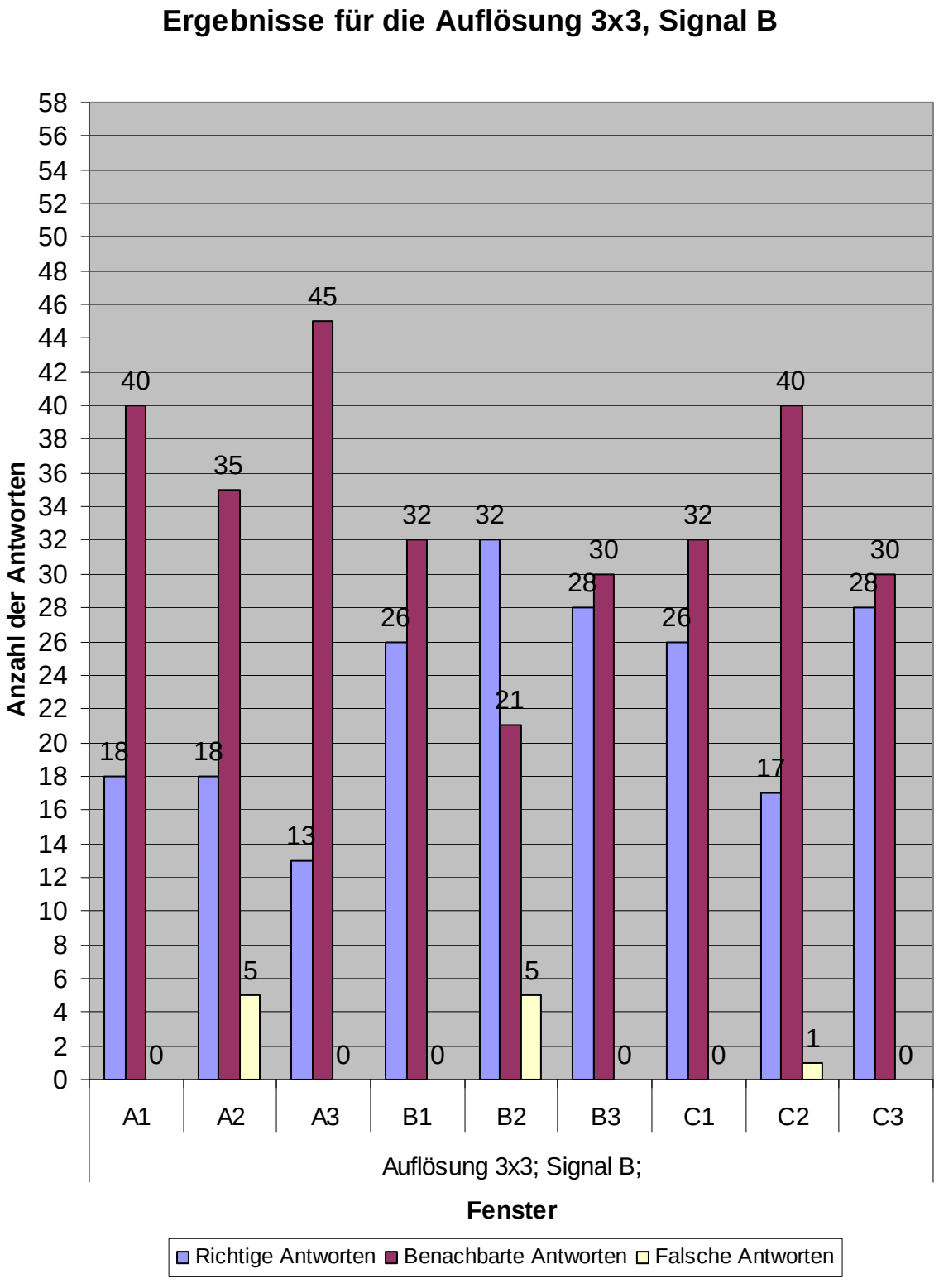


Abbildung 4.22

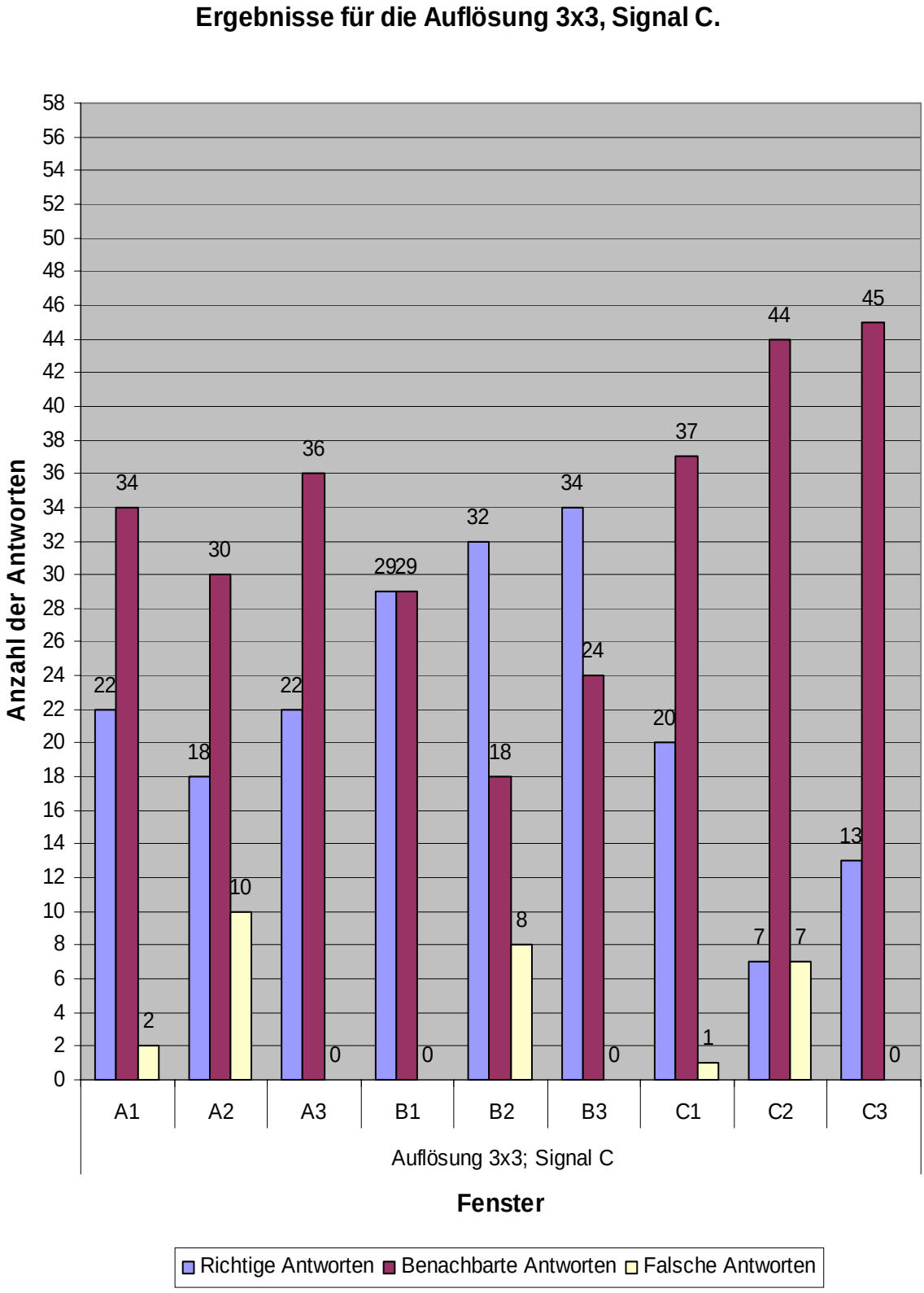


Abbildung 4.23

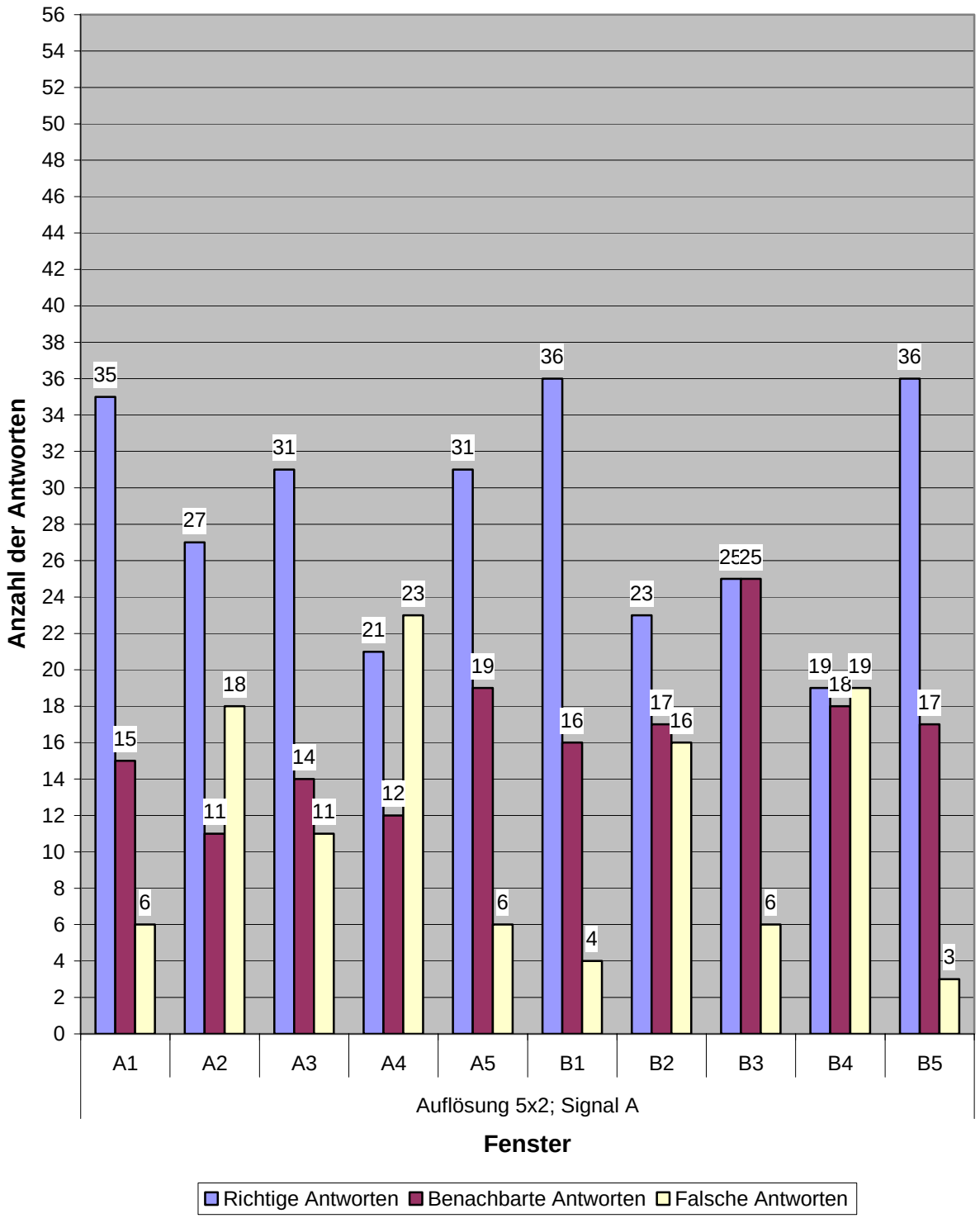


Abbildung 4.24

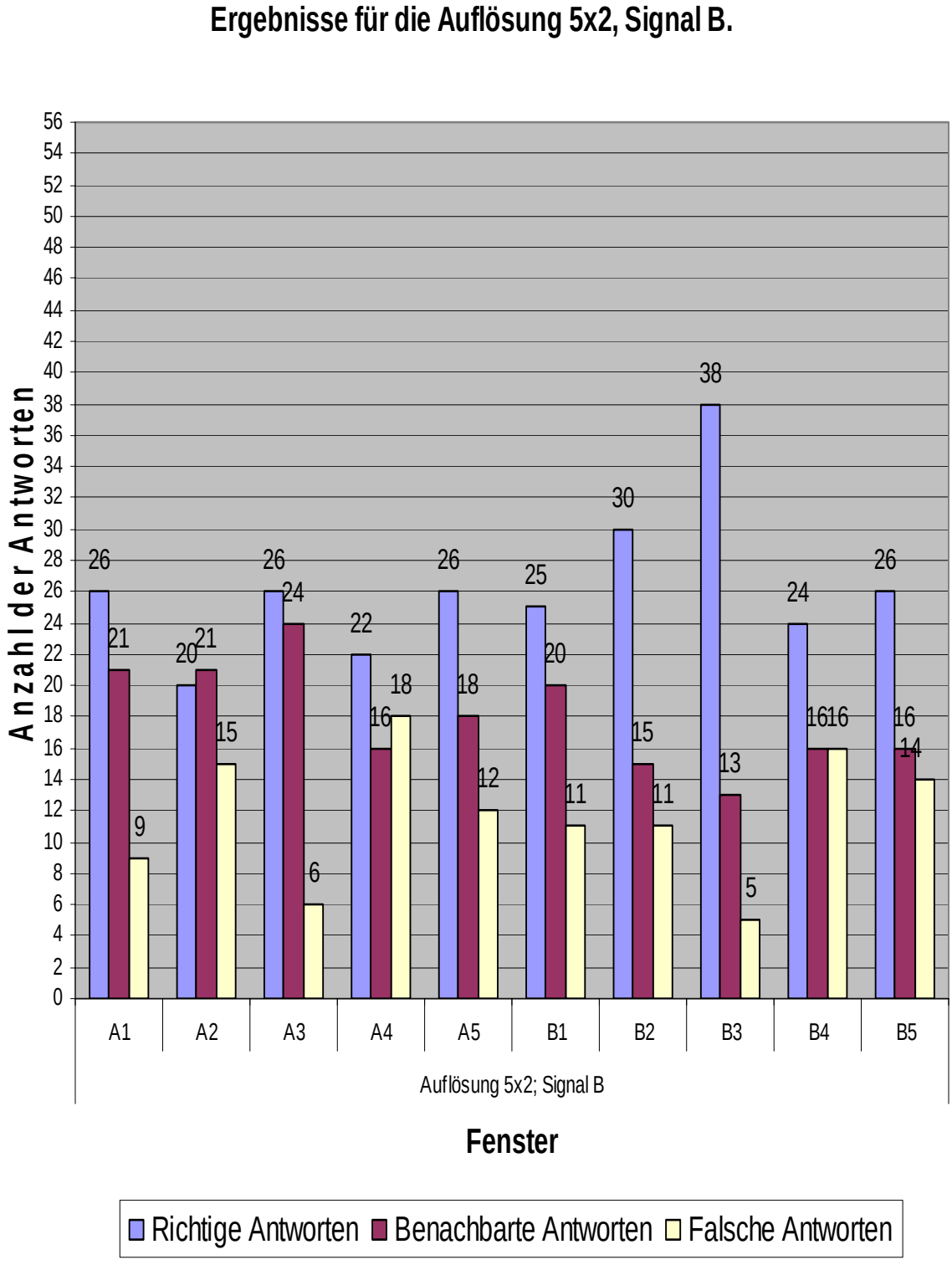


Abbildung 4.25

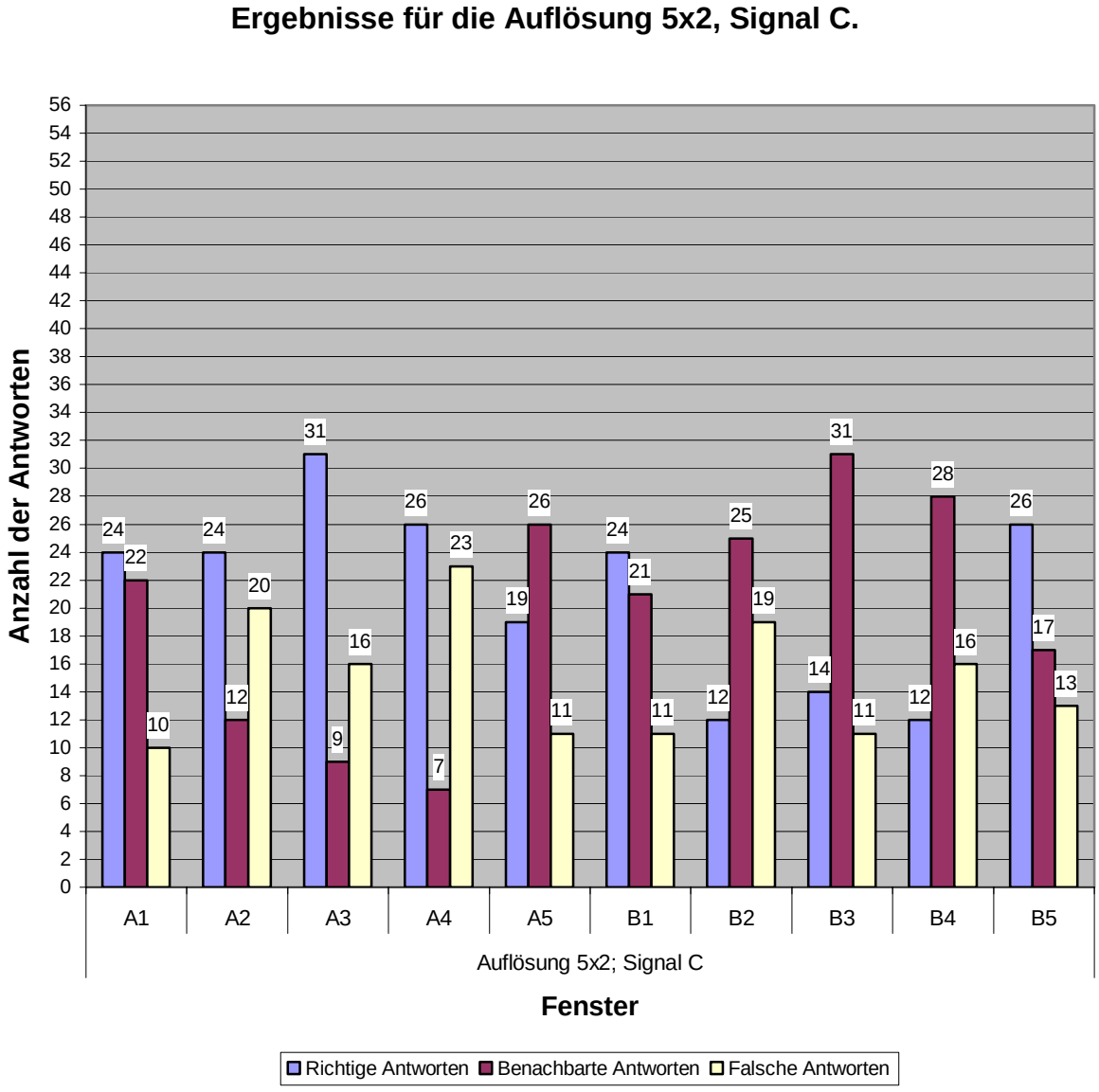


Abbildung 4.26

4.6 Diskussion der Untersuchungsergebnisse

In diesem Abschnitt werde ich nun die Untersuchungsergebnisse der 3 Untersuchungen diskutieren. Dabei werde ich auf die Diagramme sämtlicher Untersuchungen, die in dem letzten Abschnitt abgedruckt sind, eingehen und die Ergebnisse kommentieren. Zuletzt werde ich eine Schlussfolgerung ziehen und Vorschläge für zukünftige Anwendungen im Rahmen des GUIB-Projektes machen.

4.6.1 MAA-Untersuchung

Wie ich bereits erwähnt hatte, wurde bei der aller ersten Untersuchung das AVG-Netz getestet.

Ich möchte an dieser Stelle erläutern, warum es überhaupt notwendig war, das AVG-Netz zu testen, denn es ist ja selbst aus einer früheren MAA-Untersuchung hervorgegangen. Zur Erinnerung sei noch mal gesagt, dass das AVG-Netz der gerade wahrnehmbare Unterschied für Signal A darstellt. Es repräsentiert das Durchschnittsergebnis der 50 damals teilnehmenden Personen an der MAA Untersuchung.

Der erste Grund, warum das AVG-Netz nochmals getestet werden musste, lag darin, dass die Untersuchungsmethodik bei der damaligen Untersuchung von der jetzigen Untersuchungsmethodik etwas abwich. Damals hatten die Untersuchungsteilnehmer während der MAA-Untersuchung die Möglichkeit zwischen drei möglichen Antworten zu wählen (three-categorie-forced-choice), nämlich: „kein Unterschied", „unsicher" und "verschiedene Schallquellenorte" je nachdem, ob die Versuchsperson die nacheinander dargebotenen Signalimpulse mit Sicherheit, nur mit Unsicherheit oder überhaupt nicht diskriminieren konnte. Der zweite Impuls bewegte sich damals in 1° Schritten. Erstmal entfernte er sich vom Referenzpunkt bis die VP die Impulse mit Sicherheit diskriminieren konnte („Punkt der Sicherheit"). Dann kam er rückwärts auf die Referenzquelle zu bis die VP die Impulse nicht mehr diskriminieren konnte.
Zwischen diesen Punkten konnte die VP eine Unsicherheits-Region bestimmen. Als neuer Referenzpunkt wurde dann der erste Punkt festgestellt, wo die VP in beiden Bewegungsrichtungen des zweiten Impulses mit Sicherheit die Impulse diskriminieren konnte. In der Horizontalebene war der Ausgangspunkt zur Rückwertsbewegung 5°, in der Medianebene 10° weiter vom Punkt der Sicherheit entfernt.

Man sieht hier, dass die damalige Untersuchungsmethodik von der jetzigen insofern abwich, dass die Versuchsperson nur zwischen zwei Antworten zu wählen hatte, nämlich: „kein Unterschied" und „verschiedene Schallquellenorte". Außerdem bewegte sich der zweite Impuls in meiner Untersuchung nur in einer Richtung, nämlich nach rechts, links, oben oder unten und nicht wieder rückwärts. Darüber hinaus bewegte sich bei meiner Untersuchung der zweite Impuls

nicht in 1° Schritten, sondern sprang von einem Punkt innerhalb des AVG-Netzes zum nächsten Punkt innerhalb des AVG-Netzes.

Die wiederholte Überprüfung des AVG-Netzes diente also vor allem zur Überprüfung der damaligen Ergebnisse. Mein Betreuer hatte schon in meiner Aufgabenstellung die Vermutung geäußert, dass das AVG-Netz nur den BEST-CASE darstellt, denn seiner Meinung nach waren 4 vertikale und 12 Horizontale Punkte zu viel. Er wollte also mit dieser Untersuchung das so genannte „Optimale-Netz" finden, das von 90% der Versuchspersonen erkannt wurde und außerdem den WORST-CASE und BEST-Case herausfinden. Der BEST-CASE liegt vor, wenn alle Punkte erkannt wurden, der WORST-CASE repräsentiert die schlechtesten Ergebnisse.

Nun möchte ich zunächst die 90 % Grenze und den WORST- und BESTCASE für die einzelnen Signale in der Horizontal und Medianebene nennen. Da insgesamt 42 Teilnehmer an dieser Untersuchung teilnahmen, liegt die 90 % Grenze bei 37.8 Versuchsteilnehmern.

In der Horizontalebene konnten für Signal A 88.09 % der Versuchsteilnehmer mindestens 7 Punkte erkennen und 97.62% der Versuchsteilnehmer mindestens 6 Punkte erkennen. Man kann also sagen, dass bei Signal A fast 90 % der Versuchsteilnehmer zumindest 7 Punkte erkannt haben. Der BEST-CASE lag hier bei 28.57 % der Versuchsteilnehmer (= 12 Teilnehmer) und der WORST-CASE (→5 Punkte) bei nur einem Teilnehmer (=2.38%).

Bei Signal B konnten in der Horizontalebene 90.47% der Teilnehmer mindestens 7 Punkte erkennen. Der BEST-CASE lag hier bei 23.8% der Versuchsteilnehmer (= 10 Teilnehmer) und der WORST-CASE (→ 4 Punkte) bei nur einem Teilnehmer (=2.38%).

Bei Signal C konnten in der Horizontalebene 92.85 % der Teilnehmer mindestens 7 Punkte erkennen. Der BEST-CASE lag hier bei 11.9% (=5 Teilnehmer) der Versuchsteilnehmer und der WORST-CASE (4 Punkte) bei nur einem Teilnehmer (=2.38%).

In der Medianebene konnten für Signal A 95.24% der Versuchsteilnehmer mindestens 2 Punkte erkennen. Der BEST-CASE lag hier bei 54.76 % (=23 Teilnehmer) und der WORST-CASE (1 Punkt) bei 4.76 % (= 2 Teilnehmer) der Versuchsteilnehmer.

Bei Signal B konnten in der Medianebene 90.47% der Versuchsteilnehmer mindestens 1 Punkt erkennen. Der BEST-CASE lag hier bei 19.04 % (= 8 Teilnehmer) der Versuchsteilnehmer und der WORST-CASE (0 Punkte) bei 9.52% (4 Teilnehmer) der Teilnehmer.

Bei Signal C konnten in der Medianebene 88.1% der Teilnehmer mindestens 1 Punkt erkennen. Der BEST-CASE lag hier bei 14.28% (6 Teilnehmer) der Versuchsteilnehmer. Und der WORST-CASE (0 Punkte) bei 11.9 % (=5 Teilnehmern) der Versuchsteilnehmer.

Diese Ergebnisse zeigen, dass bei breitbandiger Anregung des Gehörs in der Horizontalebene recht gute Ergebnisse zustande kommen. Immerhin haben hier fast 29% der Teilnehmer alle 12 Punkte erkannt.

In der Medianebene sehen die Ergebnisse dagegen deutlich schlechter aus. Selbst bei breitbandiger Anregung konnten für Signal A nur 54 % der Teilnehmer alle 4 Punkte erkennen. Bei tiefpass gefiltertem Rauschen konnten sogar nur noch 19% der Versuchsteilnehmer alle vier Punkte unterscheiden. Am schlechtesten fiel hier das Ergebnis für hochpass gefiltertes Rauschen aus. Hier konnten nur knapp 12% der Teilnehmer alle 4 Punkte erkennen.

Richtungsabfrage

Nun werde ich die Ergebnisse der Richtungsabfrage diskutieren. Zuerst einmal will ich kurz erläutern, warum es bei der MAA Untersuchung überhaupt notwendig war, nach der Bewegungsrichtung der Schallquellen zu fragen: Die Richtungsabfrage war notwendig, um erkennen zu können, ob die Versuchspersonen mittels Lokalisation den Unterschied zwischen den Schallquellen wahrnahmen oder nur anhand einer Klangverfärbung den Unterschied zwischen den Schallquellen ausmachten.

In der Horizontalebene fielen die Ergebnisse folgendermaßen aus: Bei Signal A konnten 100% der befragten Teilnehmer die Richtung der Schallquelle richtig sagen. Bei Signal B waren es 93.3% und bei Signal C waren es 98.3 %.

In der Medianebene konnten bei Signal A nur noch 63.3% der Versuchsteilnehmer die Richtung richtig erkennen. Bei Signal B waren es noch 58.3% und bei Signal C nur noch 51.6% der Teilnehmer.

Diese Ergebnisse zeigen, dass fast alle Versuchsteilnehmer in der Horizontalebene den Unterschied zwischen den beiden Schallquellen mit Hilfe des Richtungshörens herausgefunden haben.

In der Medianebene dagegen konnten mindestens 1/3 der Leute die Bewegungsrichtung nicht korrekt erkennen. Dies lässt darauf schließen, dass in der Medianebene häufig nur (wenn überhaupt) anhand von Klangverfärbungen auf den Unterschied zwischen den Schallquellen geschlossen wurde. Diese Besonderheit ist deshalb interessant, da man damit sieht, dass die Simulation monauraler Merkmale nicht unbedingt die gewünschten Ergebnisse liefert.

Diskussion der Fehler

Die Fehler bei der MAA Untersuchung lassen sich nur psychologisch erklären. Wie ich bereits erwähnt hatte, befinden sich bei jedem Wechsel des Referenzpunktes (auch ganz zu Beginn) beim ersten Rauschburst-Paar beide Quellen am selben Ort. Damit will man testen, ob die Versuchsperson tatsächlich einen Unterschied wahrnimmt oder nur denkt, einen Unterschied wahrgenommen zu haben. Hat die Versuchsperson einen Unterschied wahrgenommen, wo in Wirklichkeit keiner war, wurde dies als Fehler gewertet.

Obwohl die Versuchsperson haargenau den gleichen Ton 2-mal hintereinander gehört hat, hat sie trotzdem einen Unterschied angezeigt. Diese Besonderheit ist für mich als Nachrichtentechniker nicht physikalisch erklärbar, sondern muss mit der Einbildung der Versuchspersonen zu tun haben.

In der Horizontalebene wurden für Signal A 30 Fehler, für Signal B 20 Fehler und für Signal C insgesamt 33 Fehler gemacht.

In der Medianebene wurden für Signal A 10 Fehler, für Signal B 11 Fehler und für Signal C 4 Fehler gemacht.

Es lässt sich hier keine genaue Tendenz erkennen, ob die Fehler eine Signalabhängigkeit oder eine Abhängigkeit zur Horizontal- und Medianebene haben. In der Medianebene wurden zwar weniger Fehler gemacht, dafür bestand die Medianebene auch aus weniger Punkten.

4.6.2 Absolute Untersuchung

Nun möchte ich abschließend die beiden absoluten Untersuchungen diskutieren. Wie ich bereits erwähnt hatte, habe ich die Antworten der Versuchspersonen nach *Richtig*, *Falsch* und *Benachbart* klassifiziert. Bei einer richtigen Antwort hat die Versuchsperson das richtige Fenster genannt und hat somit sowohl die horizontalen als auch die vertikalen Koordinaten des Fensters richtig erkannt. Bei einer benachbarten Antwort hat sie nur die horizontalen Koordinaten des Fensters richtig erkannt, die vertikalen dagegen aber nicht. Bei einer falschen Antwort hat sie entweder beide Koordinaten nicht richtig erkannt oder nur die vertikalen Koordinaten richtig erkannt. Ich habe diese Form der Klassifikation deshalb getroffen, weil ich während der absoluten Untersuchung feststellen musste, dass die Versuchspersonen sehr viel öfter vertikale Fehler machten als horizontale. Die Anzahl der benachbarten Antworten ist somit ein Maß für vertikale Fehler. Die Anzahl der falschen Antworten ist vor allem ein Maß für horizontale Fehler.

Nun möchte ich die Gesamtergebnisse der beiden Untersuchungen für alle 3 Signale in Tabellenform darstellen. Die Antworten sind in Prozentzahlen umgerechnet.

Signal A

Richtige Antworten	Benachbarte Antworten	Falsche Antworten
48.28%	47.5%	4.21%

Signal B

Richtige Antworten	Benachbarte Antworten	Falsche Antworten
39.46 %	58.43%	2.11 %

Signal C

Richtige Antworten	Benachbarte Antworten	Falsche Antworten
37.73%	56.9%	5.36%

Tabelle 1: Auflösung 3x3, Signal A, B, C;

Signal A

Richtige Antworten	Benachbarte Antworten	Falsche Antworten
50.71%	29.29%	20%

Signal B

Richtige Antworten	Benachbarte Antworten	Falsche Antworten
46.96%	32.41%	20.89%

Signal C

Richtige Antworten	Benachbarte Antworten	Falsche Antworten
37.86%	35.36%	26.79%

Tabelle 2: Auflösung 5x2, Signal A, B, C;

Wie man sieht, sind für die Auflösung 3x3 (Signal A) die richtigen und falschen Antworten etwa gleich verteilt. Bei Signal B und Signal C ist die Anzahl der benachbarten Antworten sogar größer als die der richtigen. Die Anzahl der falschen Antworten ist bei allen 3 Signalen recht gering. Dies zeigt, dass die Versuchsteilnehmer bei der Auflösung 3x3 die horizontale Richtung fast immer richtig erkannt haben, die vertikale Richtung dagegen sehr häufig nicht.

Bei der Auflösung 5x2 sind für alle 3 Signale mehr richtige Antworten gegeben worden als benachbarte. Dennoch ist die Anzahl der benachbarten Antworten für alle 3 Signale recht hoch (29%-35%), was zeigt, dass die Versuchsteilnehmer die vertikalen Koordinaten des Schallereignisses nur schwer erkennen konnten. Man muss hier bedenken, dass bei der Auflösung 5x2 das 2D VAD in nur 2 vertikale Fenster aufgeteilt war. Bei dieser Auflösung waren die Schallereignisse im richtigen und benachbarten Fenster also recht weit voneinander entfernt.

Trotzdem gelang es vielen Versuchspersonen nicht, die vertikalen Koordinaten des Schallereignisses richtig zu erkennen.

Bei der Auflösung 5x2 zeigt sich auch, dass bei allen 3 Signalen etwa 1/3 der Antworten falsch waren. Hier sieht man, dass ein nicht zu vernachlässigender Anteil der Versuchspersonen nicht mehr in der Lage war im 2D VAD fünf horizontal zueinander liegende Fenster klar voneinander zu unterscheiden.

4.6.3 Schlusswort

Am Ende möchte ich noch einpaar Empfehlungen für zukünftige Anwendungen machen. Die Untersuchungen haben gezeigt, dass die meisten Versuchsteilnehmer ohne Probleme mit einer horizontalen Auflösung von 5 umgehen können. Ich bin mir sicher, dass man geübten Personen sogar eine horizontale Auflösung von bis zu 7 zumuten kann.
Viel schwieriger wird es dagegen bei der vertikalen Auflösung. Hier hatten viele Versuchspersonen Schwierigkeiten, das richtige Fenster zu erkennen. Einem großen Teil der Versuchspersonen konnte man nicht einmal eine vertikale Auflösung von 2 zumuten. Hier zeigen sich die Schwächen einer Kopfhörersimulation. Ein anderer Grund für die schlechten Ergebnisse in der vertikalen Auflösung könnte die Tatsache sein, dass das Hören in der Medianebene ein lernabhängiger, adaptiver Vorgang ist. Die Versuchspersonen hatten vor der Untersuchung die Testsignale nie gehört, waren also völlig ungeübt und hatten somit nicht die Möglichkeit, ein „Gefühl“ für die vertikale Auflösung zu entwickeln.

Ich bin mir sicher, dass geübte Versuchspersonen in der Lage gewesen wären, zumindest mit einer vertikalen Auflösung von 2, vielleicht aber auch 3 umzugehen.

Für Zukünftige Anwendungen würde ich daher raten, dass 2D VAD mit einer Auflösung von 5x2 zu betreiben.

Vielleicht sollte man auch einigen Versuchspersonen die Möglichkeit geben, einige Zeit lang die Testsignale zu hören und anschließen mit diesen geübten Versuchspersonen die Untersuchung mit einer 5x3 Auflösung wiederholen. Ich bin mir sicher, dass dann die Ergebnisse besser ausfallen würden.

5 Literaturverzeichnis

[Bla83] Jens Blauert, Räumliches Hören, Springer Verlag 1983

[Art-diss] Dissertation von Frank Artinger, Entwurf und Erprobung eines PC-basierten Systems zur Diagnose und Therapie von Störungen der lexikalischen Sprachverarbeitung, Fakultät der Elektrotechnik der Universität Bundeswehr

[Wers-diss] Dissertation von Wersényi György, HRTFs in Human Localization, Brandenburgische Technische Universität Cottbus

[MG91] J., C. Middlebrooks and D., M. Green. Sound localization by human listeners. Annual Reviews of Psychology, 42:135_159, 1991.

[WADW93] E., M. Wenzel, M. Arruda, Kistler D., J., and F,. L. Wightman. Localization using nonindividualized head-related transfer functions. Journal of the Acoustical Society of America, 94:111_123, 1993.

[Btrn] Dokumentation der Beachtron Karte, The Beachtron, Crystal River Engineering, Inc.

Im zweiten Kapitel dieser Arbeit habe ich viele Abbildungen benutzt, die ich von der freien Enzyklopädie **Wikipedia** heruntergeladen habe (www.wikipedia.de).

Darüber hinaus habe ich im Rahmen dieser Arbeit auch einige Kapitel aus dem folgenden Buch studiert, aus der ich jedoch nicht zitiert habe: Das Ohr als Nachrichtenempfänger, Feldtkeller, Zwicker, Springer Verlag.

6 Anhang

6.1 Anhang A

6.1.1 Bedienungsanleitung der beiden Programme

In diesem Anschnitt befindet sich eine kurze Bedienungsanleitung der beiden Programme, die ich geschrieben habe, um die MAA-Untersuchung und die absolute Untersuchung durchzuführen.

Bedienungsanleitung für das erste Programm (MAA Untersuchung)

Wenn man das erste Programm startet, muss man zuerst einige Parameter einstellen, bevor die eigentliche Untersuchung beginnen kann.

Startet man das Programm für die MAA Untersuchung, wird man zuerst gefragt, ob man den Abstand der beiden Ohren angeben will. Man hat bei dieser Frage die Möglichkeit mit Ja (durch Drücken der Taste „y“) oder mit Nein (durch Drücken der Taste „n“) zu antworten. Wenn man mit Nein antwortet, wird der default Wert von 6.0 Zoll (15.24 cm) für den Abstand der Ohren verwendet. Wenn man mit Ja antwortet, kann man den Ohrabstand manuell eingeben. Dabei muss der eingegebene Wert zwischen 15 cm – 50 cm liegen. Gibt man aus versehen einen Wert ein, der außerhalb der Grenzen liegt, wird der Anwender aufgefordert einen Wert einzugeben, der innerhalb der oben genannten Grenzen liegt.

Danach wird man gefragt, ob man eine Auflösung für die MAA Untersuchung eingeben will. Beantwortet man diese Frage mit Nein (durch Drücken der Taste „n“), wird die default Auflösung für die MAA Untersuchung verwendet. Mit default Auflösung ist hier das AVG-Netz gemeint. D.h. die sich bewegende Schallquelle würde hier von einem Punkt innerhalb des AVG-Netzes zum anderen Punkt springen.
Beantwortet man die Frage mit Ja (durch Drücken der Taste „y“), hat man im Folgenden die Möglichkeit die Auflösung für die X- und Y-Achse manuell einzugeben. Der eingegebene Wert für die Auflösung darf nicht kleiner als 1° sein. Gibt man trotzdem einen kleineren Wert ein, wird man aufgefordert einen richtigen Wert für die Auflösung einzugeben.

In der nächsten Frage wird man gefragt, ob man eine Verstärkung für Signal A eingeben will. Wird diese Frage mit Nein beantwortet, wird die default Verstärkung (0 dB) für Signal A verwendet. Wird die Frage aber mit Ja beantwortet, kann man manuell einen Wert für die Verstärkung (in dB) des Signal A eingeben. Dabei ist Signal B immer um +10 dB und Signal C immer um +6 dB stärker als Signal A. Das Programm akzeptiert nur positive Werte für die Verstärkung.

Danach wird der Anwender aufgefordert die Initialen der Versuchsperson (bestehend aus 3 Buchstaben) einzugeben. Später wird die Textdatei, die die Untersuchungsergebnisse der Versuchsperson enthält, nach den Initialen der Versuchsperson benannt.

Als nächstes wird man aufgefordert, für die MAA Untersuchung eines der 3 Standard Signale (Signal A, Signal B, Signal C) oder ein beliebiges anderes Signal (Signal D) auszuwählen (durch Drücken der Tasten „a“, „b“, „c“, „d“).
Wählt man Signal D für die Untersuchung aus, sucht das Programm im aktuellen Verzeichnis nach der Datei Signal.wav. Mann muss also diejenige Wave-Datei, die man für die Untersuchung verwenden möchte, umbenennen in Signal.wav.

Der Anwender wird dann aufgefordert durch Drücken der Tasten „u“, „d“, „r“, „l“
die Bewegungsrichtung der Schallquelle zu bestimmen. Hat man eine Richtung ausgewählt, muss man nur noch die Eingabetaste drücken, um die Untersuchung zu starten.

Während der Untersuchung kann man die Position der Referenzschallquelle verändern, indem man die Leertaste drückt. Hat man während der Untersuchung das Ende der Achse erreicht, wird der Anwender darauf aufmerksam gemacht. In diesem Falle wird der Anwender gefragt, ob er die Untersuchung in eine andere Richtung fortsetzen möchte. Er kann hier mit Ja oder mit Nein antworten. Antwortet er mit Nein, werden die bis dahin gesammelten Untersuchungsergebnisse aus dem Arbeitsspeicher in die Textdatei übertragen und abgespeichert. Der Name der Textdatei ist aus 5 Buchstaben zusammengesetzt. Die ersten 3 Buchstaben bestehen aus den Initialen der Versuchspersonen dann folgt ein Bindestrich. Danach folgt entweder der Buchstabe A, B, C, oder D, je nachdem, welche Schallquelle für die Untersuchung benutzt wurde.

Wenn dieselbe Achse während der Untersuchung mehrere male durchlaufen wurde, wird in der Textdatei nur der letzte Durchlauf abgespeichert.

Bedienungsanleitung für das zweite Programm (absolute Untersuchung)

Wie beim ersten Programm, muss man zu Beginn des zweiten Programms auch einige Parameter einstellen, bevor die eigentliche Untersuchung beginnen kann.

Startet man das Programm, wird man zuerst gefragt, ob man den Abstand der beiden Ohren angeben will. Man hat bei dieser Frage die Möglichkeit mit Ja (durch Drücken der Taste „y“) oder mit Nein (durch Drücken der Taste „n“) zu antworten. Wenn man mit Nein antwortet, wird der default Wert von 6.0 Zoll (15.24 cm) für den Abstand der Ohren verwendet. Wenn man mit Ja antwortet, kann man den Ohrabstand manuell eingeben. Dabei muss der eingegebene Wert zwischen 15 cm – 50 cm liegen. Gibt man ausversehen einen Wert ein, der außerhalb der Grenzen liegt, wird der Anwender aufgefordert einen Wert einzugeben, der innerhalb der oben genannten Grenzen liegt.

Danach wird der Anwender gefragt, in wie viele vertikale Fenster das 2D VAD aufgeteilt werden soll. Er hat hier die Möglichkeit, zwischen 1-5 Fenster zu wählen. Wenn man einen unzulässigen Wert eingibt, wird der Benutzer darauf aufmerksam gemacht und muss erneut einen Wert eingeben. Hat man die Anzahl der vertikalen Fenster eingegeben wird in der nächsten Frage nach der Anzahl der horizontalen Fenster gefragt. Hier hat der Benutzer die Möglichkeit zwischen 3-7 Fenstern zu wählen. Wird hier ein unzulässiger Wert eingegeben, wird man wieder darauf aufmerksam gemacht und muss einen Wert eingeben, der innerhalb der Grenzen liegt.

In der nächsten Frage wird man gefragt, ob man eine Verstärkung für Signal A eingeben will. Wird diese Frage mit Nein beantwortet, wird die default Verstärkung (0 dB) für Signal A verwendet. Wird die Frage aber mit Ja beantwortet, kann man manuell einen Wert für die Verstärkung (in dB) des Signal A eingeben. Dabei ist Signal B immer um +10 dB und Signal C immer um +6 dB stärker als Signal A. Das Programm akzeptiert nur positive Werte für die Verstärkung.

Für den Fall, dass der Anwender sich für eine vertikale Auflösung von 2 oder 3 entscheidet, bekommt er die Möglichkeit den folgenden zusätzlichen Parameter zu setzen: Er kann wählen, ob er für die zwei bzw. drei vertikalen Positionen jeweils unterschiedliche Schallquellen benutzt oder für jede Position innerhalb des 2D VAD die gleiche Schallquelle benutzt. Entscheidet er sich für unterschiedliche Schallquellen, so wird bei einer vertikalen Auflösung von 2 für sämtliche Fenster, die oben liegen, hochpassgefiltertes Rauschen benutzt und für sämtliche Fenster, die unten liegen tiefpassgefiltertes Rauschen benutzt. Bei einer vertikalen Auflösung von 3 würde bei unterschiedlichen Schallquellen für die oben gelegenen Fenster hochpassgefiltertes Rauschen benutzt, für die in der Mitte gelegenen weisses Rauschen und für die unten gelegenen tiefpassgefiltertes Rauschen.
Dieser Parameter sollte dann gesetzt werden, wenn man möchte, dass die Versuchspersonen die horizontale Richtung des Schallereignisses mit Hilfe ihres natürlichen Richtungshörens herausfinden, die vertikale Richtung dagegen nur anhand der Klangunterschiede der verschiedenen Rauscharten. (für meine eigenen Untersuchungen habe ich diesen Parameter nicht gesetzt und für alle Fenster die gleiche Schallquelle benutzt).
Als nächstes wird man aufgefordert, für die Untersuchung eines der 3 Standard Signale (Signal A, Signal B, Signal C) oder ein beliebiges anderes Signal (Signal D) auszuwählen (durch Drücken der Tasten „a“, „b“, „c“, „d“).
Wählt man Signal D für die Untersuchung aus, sucht das Programm im aktuellen Verzeichnis nach der Datei Signal.wav. Mann muss also diejenige Wave-Datei, die man für die Untersuchung verwenden möchte, umbenennen in Signal.wav.

Als nächstes wird man aufgefordert die Dauer der Pause zwischen den Schallsignalen einzugeben (zwischen 0-7000 ms). Wenn man 0 Sekunden eingibt, wird das Signal solange ununterbrochen abgespielt, bis man eine Taste drückt. Bei einer Pause größer als 0 ms wird die Wavedatei einmal abgespielt, dann folgt die vom Benutzer festgelegte Pause, danach wird die Wavedatei erneut abgespielt usw. (je nach Anzahl der Wiederholungen).

Danach wird der Anwender gefragt, wie oft die Schallquelle innerhalb desselben Fensters wiederholt werden soll. Man kann hier eine Zahl zwischen 1-20 eingeben.

Als nächstes wird der Anwender gefragt, ob die Fensterauswahl automatisch erfolgen soll (mittels eines Zufallgenerators) oder ob der Anwender die Fenster selbst auswählen will. Hier hat man die Möglichkeit mit Ja oder Nein zu antworten.

Hat man die oben beschriebenen Parameter eingestellt, kann die eigentliche Untersuchung beginnen.

Hat der Anwender sich für eine automatische Fensterauswahl entschieden, wird zuerst vom Programm zufällig ein Fenster ausgewählt, aus dem die Schallquelle ertönt. Die Versuchsperson muss dem Versuchsleiter dann sagen, aus welchem Fenster sie die Schallquelle gehört hat. Das

Programm wartet solange, bis der Versuchsleiter die Antwort des Teilnehmers eingegeben hat und die Eingabetaste gedrückt hat. Hat der Versuchsleiter eine gültige Fensternummer eingegeben, fragt das Programm, ob man mit der Untersuchung fortfahren möchte oder mit der Taste ESC das Programm verlassen möchte.
Wenn der Versuchsleiter sich aber für eine manuelle Fensterauswahl entscheidet, muss er zuerst die Fensternummer eingeben, aus dem die Schallquelle ertönen soll. Nach der Antwort der Versuchsperson muss er natürlich auch die Antwort eingeben.
Wenn schließlich irgendwann die Escape Taste gedrückt wird, wird das Programm nicht sofort verlassen, sondern es wird noch abschließend nach den Initialen der Versuchsperson gefragt. Diese werden benutzt, um die Textdatei mit den Untersuchungsergebnissen anzulegen. Die Textdatei besteht hier ebenfalls aus fünf Buchstaben. Die ersten 3 Buchstaben sind die Initialen, dann folgt ein Bindestrich. Am Ende folgt der Buchstabe A, B, C, oder D, je nachdem, welche Schallquelle für die Untersuchung benutzt wurde.

6.2 Anhang B

6.2.1 Funktionen zur Steuerung der Beachtron Karte

Um die Beachtron Karte steuern zu können, hat der Hersteller dieser Karte fünfzehn C-Funktionen in Form von einer Software Bibliothek mitgeliefert. Ich selber habe für meine beiden Programme nur 12 von 15 Funktionen benötigt. In der Dokumentation der Beachtron Karte befindet sich eine ausführliche Beschreibung sämtlicher Funktionen. Im Folgenden sind die Beschreibungen der 12 Funktionen, die ich für meine Programme benötigt habe, abgedruckt. Da die Dokumentation in Englisch geschrieben wurde, sind die jetzt folgenden Funktionsbeschreibungen auch in Englisch.

Software interface

The function calls that allow a programmer to interact with the Beachtron can be grouped into five categories:

- **Utility** functions that initialize, update, and close the Beachtron, and provide error management:

```
int   btron_init           (int sources, int mode);
int   btron_update_audio   (void);
int   btron_close          (void);
int   btron_error          (int idx);
```

- Functions which allow for the definition and positioning of the **listener's head:**

```
int   btron_model_head     (const float *ear_offset, const char *ahm);
int   btron_locate_head    (const float *head);
```

- Functions which allow for the definition, positioning, and amplification of **sound sources:**

```
int   btron_define_source (int id, int pts, const float *table);
int   btron_locate_source (int id, const float *sloc);
int   btron_amplfy_source (int id, float dB);
```

- **MIDI-related** functions, which allow for interaction with the Proteus sound synthesizer in the form of MIDI streams and commands:

```
int   btron_send_midi (int id, const unsigned char *midistr);
int   btron_set_midi  (int id, int cmd, void *data);
int   btron_msg_midi  (int id, int msg, int chnl, int data1, int data2);
```

- Functions for opening, playing, and closing **sampled soundfiles** (waveforms):

```
const wavFt *
      btron_open_wave  (const char *wavefile, int mode);
int   btron_ctrl_wave  (int id, const wavFt *wave, int cmd, void *data);
int   btron_close_wave (const wavFt *wave);
```

These functions are described in detail in the "Reference" section below (in this chapter). Also see "Data structures" for a description of the `wavFt` data type.

With the exception of the `btron_open_wave()` function (which returns a `NULL` pointer if an error occurs), all Beachtron library routines return an `int` status code. Successful operation of a function is normally indicated by the `Ok` result code, while errors are denoted by codes `Error0`, `Error1`, ...* The specific error codes and messages are described for each function call (see "Return Value") in the "Reference" section.

* The result codes `Ok`, `Error0`, `Error1`, ... are macros corresponding to integers 1, 0, -1, etc.

btron_amplfy_source

Synopsis

```
#include <btron.h>
int  btron_amplfy_source (int id, float dB);
```

Description

amplifies or attenuates the dynamic range over distance of the indicated sound source *id* by *dB* decibels. The default source gain (0 dB) is set so that maximum possible volume is reached at a distance of 2.5 units from an ear. Distant sound sources may need to be set much higher (as much as +30 dB), in order to be audible at the listener's position. Note that very high gain sources will cease to get louder as they approach the listener within a certain range, when the maximum signal level is reached. This range in which *volume* "clipping" occurs, normally 2.5 inches for a source with the default 0-dB amplification, may be quite large for sources with strong amplification.

Parameters

id	the index (from zero) of the sound source in reference. The macro `ALL_SOURCES` is supported.
dB	the amplification in decibels. The macro `GAIN_dB_OFF` is provided in BTRON.H to definitively shut off a sound source. Any value less than -120 dB also is interpreted as off.

Return Value

`Ok`	on success.
`Error0`	on source *id* out of range, or if no Beachtron sources have been initialized.

Example

```
btron_amplfy_source(ALL_SOURCES, GAIN_dB_OFF);
```

btron_close

Synopsis

```
#include <btron.h>
int  btron_close (void);
```

Description deallocates host and DSP resources (which may then be reallocated with another `btron_init()` call). `btron_close()` will gently shut off all audio, unless initialization (previously) specified `_VERBOSE_` mode. `btron_close()` is required to safely terminate a host application.

Parameters none

Return Value `Ok` on completion.

Example

```
btron_close();
```

btron_close_wave

Synopsis

```
#include <btron.h>
int  btron_close_wave (const wavFt *wave);
```

Description

closes the wave, and frees the signal and the wave structure. If *wave* is attached to a sound source and is playing, it will be stopped before the wave is closed. In order to properly deallocate resources, each (successful) call to `btron_open_wave()` must be balanced with a call to this routine.

Parameters

wave	the wave structure to be closed.

Return Value

`Ok`	on success.
`Error1`	on invalid wave structure.

Example

```
const wavFt *wavep = btron_open_wave("TEST.WAV",4);
btron_close_wave(wavep);
```

btron_ctrl_wave

Synopsis

```
#include <btron.h>
int  btron_ctrl_wave
       (int id, const wavFt *wave, int cmd, void *data);
```

Description

sends a command *cmd*, relating to the waveform *wave* and/or source *id*, to the DSP associated with source *id*.

If the waveform *wave* is already attached to some DSP, `btron_ctrl_wave()` will report a DSP conflict (`Error3`). If the DSP is already playing a waveform, `btron_ctrl_wave()` will report a DSP error (`Error4`). While it can spatialize two sources, each DSP can only play back one monophonic waveform at a time. Hence playing or looping a waveform to one source will prohibit its paired source from playing a waveform until the currently playing waveform is completed or stopped.

For further information above waveform playback, refer to the discussion "Sound files" earlier in this chapter.

Parameters

id — the index (from zero) of the sound source in reference. The macro `ALL_SOURCES` is *not* supported.

wave — a pointer to the waveform structure affected by all commands except WaveCTRL_STOP.

cmd — an integer which specifies one of a number of commands (as enumerated in BTRON.H):

`WaveCTRL_RWND`	resets the position of waveform *wave* to the beginning
`WaveCTRL_STRT`	resets and plays waveform *wave* from the beginning
`WaveCTRL_PLAY`	plays waveform *wave* from current position
`WaveCTRL_STOP`	stops playing waveform *wave*, maintaining current position
`WaveCTRL_LOOP`	plays waveform *wave* from the current position with loop flag enabled. When playback reaches the end of the sound file, the file is automatically rewound to the beginning. This process continues until the playback is stopped (`WaveCTRL_STOP`) or the loop flag becomes disabled (`WaveCTRL_NOLP`).
`WaveCTRL_NOLP`	disables loop flag for waveform *wave*

data — Presently ignored. For future compatibility, it is recommended that `NULL` be specified for this parameter.

Return Value

`Ok`	on success.
`Error0`	on source *id* out of range, or if no Beachtron sources have been initialized, or if waveform-playback interrupts are disabled (presumably via BTRNADRS=0).
`Error1`	on an illegal waveform structure *wave*.
`Error2`	on an illegal command *cmd*.
`Error3`	on a DSP conflict, i.e., *wave* already playing on some DSP.
`Error4`	on a DSP error, i.e., failure to perform command.

Example

```
const wavFt *wavep = btron_open_wave("TEST.WAV", 4);
btron_ctrl_wave(0, wavep, WaveCTRL_LOOP, NULL);
btron_close_wave(wavep);
```

btron_define_source

Synopsis

```
#include <btron.h>
int  btron_define_source (int id, int pts, const float *table);
```

Description

defines an audio source's radiation pattern about its boresight axis. The radiation profile of a sound source *id* is specified with a table of relative sound pressure levels in decibels, sampled equi-angularly at *pts* points from the boresight N-pole to S-pole, inclusive. The boresight direction is defined as a source's positive roll axis. All defineable radiation patterns are symmetric about the roll axis (i.e., no rectangular horn speakers). For off-axis angles from source to listener that fall between sample points, the profile is linearly interpolated. Designed for empirically sampled data, `btron_define_source()` also provides effective profile definition with very few artificial points.

Uniformly radiating sources save considerable computation and are specially defined by *pts* = zero. In the special case of *pts* = 1, the boresight direction is set to the value pointed to by table, while the opposite pole is defined by (*table + GAIN_dB_OFF)/2. If a NULL table pointer is given, the profile is set to the first *pts* points of the existing table. The maximum table size supported is specified by the macro MAX_RADPROFILE.

Parameters

id	the index (from zero) of the audio source in reference. The macro ALL_SOURCES is supported.
pts	the number of points to be used from the radiation profile table. If zero, the source is specially defined as uniform. A range error occurs if *pts* is negative or greater than MAX_RADPROFILE.
table	a pointer to at least *pts* radiation data points in dB. An undetectable error may occur if *pts* is larger than the number of points actually in the table. If NULL, the first *pts* points of the existing profile table will be used.

Return Value

Ok	on success.
Error0	on source *id* out of range, or if no Beachtron sources have been initialized.
Error1	on *pts* out of range.
Error2	on internal table allocation failure.

Example

```
const float radiation_table[4] = { 0.0, -5.0, -20.0, -60.0 };
btron_define_source(mysource, 4, radiation_table);
```

Remarks

A positive value for *pts* requires that the source *orientation* be maintained as well as translation. Hence, all subsequent calls to `btron_locate_source()`, regarding the same source *id*, must specify yaw and pitch. However, roll orientation is still ignored, because the radiation pattern is symmetrical about the roll axis.

btron_error

Synopsis

```
#include <btron.h>
int  btron_error (int idx);
```

Description

primarily used for remote customer-to-factory debugging. Users that have trouble initializing their hardware may need to provide these values when calling the factory for troubleshooting.

For technical support and related assistance, see "Problems" at the end of Chapter 2.

Parameters

idx	the index (from zero to `ErrorLAST`) of the error value in reference.

Return Value

`Error1`	on an index out of range.

Otherwise, returns the error code at the referenced error index *idx*.

Example

```
if (btron_init(1,_VERBOSE_) < Ok) {
  for (i=0; i<ErrorLAST; i++)
    printf("\n Error#%d: %0x",btron_error(i));
  abort();
}
```

btron_init

Synopsis

```
#include <btron.h>
int  btron_init (int sources, int mode);
```

Description

computes, detects, and allocates hardware resources (i.e., Beachtrons) to provide the services for the requested number of sources. The allocated DSP hardware is reset, and the DSP driver program is downloaded. All memory is suitably initialized. The listener's head is located at the origin. All sound sources are initially positioned 50 inches in front of the listener's head. The Proteus synthesizer is initialized with MIDI channels 0–7 panned to the input channels of *even*-numbered sources, and MIDI channels 8–15 to the input channels of *odd*-numbered sources. Each MIDI channel is patched to the Proteus preset instrument which corresponds to the channel number (i.e., channel 0 = Stereo Piano, channel 1 = Hall strings, ...).

Parameters

sources — total number of sources heard by the listener's head.

mode — bit field "ORed" from BTRON.H macros and enums:

units—select *one* from enum list:

```
enum  ATRNunits {
  AtrnINCHES,
  Atrn_FEET_,
  AtrnMMETER,
  AtrnCMETER,
  Atrn_METER,
};
```

`_VERBOSE_` — select verbose mode (writes progress information to standard output)

`_SHOWSET_` — select detect-and-exit mode

Return Value

`Error0` — if fewer than 1 sources are requested or accessible.

`Error1` — on allocation error or DSP binary load failure. Specific error information may be retrieved with `btron_error()`.

`Error2` — on Acoustic Head Map load failure.

Otherwise, returns the number of Beachtrons allocated—should be (sources + 1)/2, if sufficient hardware is available.

Example

```
if (btron_init(srcs, (AtrnINCHES | _VERBOSE_)) < Ok)
  abort();
```

Remarks

A Beachtron is initialized once, for a single listener. After a call to `btron_init()`, all subsequent calls (to `btron_init()`) are ignored until a `btron_close()` has been performed. (In an Acoustetron environment, Beachtron cards can be dynamically initialized for multiple listeners).

btron_locate_head

Synopsis

```
#include <btron.h>
int  btron_locate_head (const float *head);
```

Description

locates the head of the listener six dimensionally in world coordinates. It only updates what has changed, recalculating pinnae location as needed.

Parameters

head — an array of six `float`s, in order as follows:

`AtrnX`	world *x*-axis coordinate.
`AtrnY`	world *y*-axis coordinate.
`AtrnZ`	world *z*-axis coordinate.
`AtrnYAW`	angle on $-\pi$ to π from the world *x*-axis about the world *z*-axis of the projection of the head's *x*-axis onto the world *x*-*y* plane. Looking down at the *x*-*y* plane a counter-clockwise rotation is positive.
`AtrnPTC`	angle on $-\pi/2$ to $\pi/2$ from the world *x*-*y* plane of the head's *x*-axis about the world *y*-axis. *Remember* that with *x* forward and *z* up, a positive pitch is down!
`AtrnROL`	angle on $-\pi$ to π from the world *y*-axis about the world *x*-axis of the head's *y*-axis. From the listener's point of view, a clockwise roll of the head, rolls *y* into *z* and is therefore positive.

Return Value

`Ok`	on completion.
`Error0`	if no Beachtron sources have been initialized.
`Error1`	if *head* is `NULL`.

Example

```
const float headloc[6] = { 10., 20., 30., 0., 0., 0. };
btron_locate_head(headloc);
```

btron_locate_source

Synopsis

```
#include <btron.h>
int  btron_locate_source (int id, const float *sloc);
```

Description

locates an audio source *id* six dimensionally in world coordinates.

Parameters

id — the index (from zero) of the audio source in reference. The macro `ALL_SOURCES` is *not* supported. To co-locate audio sources, one must manually call this function for each of the coincident sounds.

sloc — the source location—an array of six `floats`, in order as follows:

`AtrnX`	world *x*-axis coordinate.
`AtrnY`	world *y*-axis coordinate.
`AtrnZ`	world *z*-axis coordinate.
`AtrnYAW`	angle on $-\pi$ to π from the world *x*-axis about the world *z*-axis of the projection of the source's *x*-axis onto the world *x*-*y* plane. Looking down at the *x*-*y* plane, a counter-clockwise rotation is positive.
`AtrnPTC`	angle on $-\pi/2$ to $\pi/2$ from the world *x*-*y* plane of the source's *x*-axis about the world *y*-axis. *Remember* that with *x* forward and *z* up, a positive pitch is down!
`AtrnROL`	angle on $-\pi$ to π from the world *y*-axis about the world *x*-axis of the source's *y*-axis. From the source's point of view, a clockwise roll of the sound rolls *y* into *z* and is therefore positive.

Return Value

`Ok`	on success.
`Error0`	on source *id* out of range, or if no Beachtron sources have been initialized.
`Error1`	if *sloc* is NULL.

Example

```
const float srcloc[6] = { -10., -20., 0., 0., 0., 0. };
btron_locate_source(mysrc, srcloc);
```

Remarks

Only the first three `floats` are used when the audio source is in uniform radiation mode. However, as a safe programming practice, you should always maintain pointers to six-element data structures. `AtrnROL` is not presently used since non-uniform radiation is currently symmetric about the boresight roll axis, but it may be used in the future.

btron_model_head

Synopsis

```
#include <btron.h>
int  btron_model_head
       (const float *ear_offset, const char *ahm);
```

Description assists in modelling the specifics of the listener (head size and pinnae characteristics) and listener's apparatus (head tracker).

Parameters

ear_offset an array of three `floats` offsetting the pinnae from the six-degree-of-freedom head locations given with `btron_locate_head()`. The offsets specify a translation in head coordinates (see Figure 5, earlier in this chapter) *from* the given head point *to* the pinnae axis, in the following order:

- *Aural-Ocular* is along the *x*-axis (positive out through the nose), and is typically zero or an offset from ocular axis coordinates. The ocular axis is in front of the aural axis, so such an offset would be negative. *Aural-Ocular* is initialized to zero.
- *Inter-Aural* is along the *y*-axis (positive out through the left ear), and *must* be specified in a *full* width of the pinnae. While an HRTF pair contains the information about the time delay owing to head size, some focusing of the distance cues may be achieved with fine interaural adjustments. *Inter-Aural* must be positive and is initialized to the INTERAURAL width constant, defined in BTRON.H.
- *Aural-Crown* is along the *z*-axis (positive up out the top of the head), and is typically either the vertical separation of ocular and aural axes, or is the vertical offset to the head tracking sensor, often on top of the head. The head crown is above the aural axis, so such an offset would be negative. *Aural-Crown* is initialized to zero.

ahm If *ahm* points to a valid HRTF filename, the HRTF map will be loaded onto all initialized Beachtrons. If the first character in the *ahm* string is '*', the default HRTF map will be searched for and, if found, re-loaded to all initialized Beachtrons. If *ahm* is NULL the current HRTF will not be affected.

Return Value

`Ok`	on success.
`Error0`	if no Beachtron sources have been initialized.
`Error1`	if *`ear_offset`* is `NULL`.
`Error2`	on failure to load HRTF map file.

Example

```
const float offsets[3] = { 0.0, INTERAURAL, CROWN_OFFSET };
btron_model_head(offsets, NULL);
```

(`CROWN_OFFSET`, defined in BTRON.H, represents a typical crown-to-aural axis offset.)

btron_open_wave

Synopsis

```
#include <btron.h>
const wavFt *btron_open_wave (const char *wavefile, int mode);
```

Description

opens a sound file referred to by the filename *wavefile* from disk, returning a pointer to the associated wavefile structure (`wavFt`). `btron_open_wave()` can be called to open one or more sound files, independently of, i.e., before or after, the call to `btron_init()`. Playback is initiated through the routine `btron_ctrl_wave()`. The wavefile may be closed, freeing all allocated memory, using `btron_close_wave()`.

For further information above waveform playback, refer to the discussion "Sound files" earlier in this chapter.

Parameters

wavefile	a string which specifies the filename to be loaded from disk. If the filename ends with ".WAV", the file must be formatted as a Windows WAVE sound file. Otherwise, the file is played assuming it contains 16-bit monophonic samples, with no file header.
mode	the amount of memory that is being allocated for the file in (32-Kbyte chunks) * *mode*. For example, a *mode* value of 4 would cause `btron_open_wave()` to allocate 128 Kbytes. Mode 0 will allocate the maximum amount of memory possible. If there isn't enough memory allocated, the waveform will be played from disk.

Return Value

`wavFt*`	on success. The returned structure will point to a completely specified wavefile structure (as defined in BTRON.H).
`NULL`	on illegal wavefile filename. `btron_open_wave()` will write error messages related to corrupt wave data to standard error.

Example

```
char *fname = "test.wav";
const wavFt *wave = btron_open_wave(fname, 4);
if (wave == NULL)
  printf("%s failed to open.\n",fname);
```

btron_update_audio

Synopsis

```
#include <btron.h>
int  btron_update_audio (void);
```

Description

Synchronizes and controls signal processing. This routine checks all data structures for updates since the previous call, recomputes convolution parameters (such as relative source-to-head position and source audibility to each ear) for all affected source-to-head relationships, and finally updates all the Beachtron controls in DSP memory with any revised data. Should be called *once* every time you want the audio updated. It may be called often, as it returns quickly without disturbing the DSP(s) if there is nothing to be done.

This routine also performs a side duty of providing support services for disk-based waveform playback, i.e., reading the new sections of a waveform from disk into RAM buffers, awaiting playback at interrupt-level. This routine *must* be called occasionally (at least every few hundred milliseconds) to assure continuous playback of long waveforms.

Parameters

none

Return Value

`Ok`	on success, or if all Beachtrons are already up-to-date.
`Error0`	if no Beachtrons have been initialized.
`Error1`	if one or more Beachtrons could not be interrupted to perform an update.

Example

```
btron_update_audio();
```

Remarks

To maintain synchronization of audio processing in a system in which more than one object (listener and/or sources) are moving simultaneously, you should complete all necessary calls to `btron_locate_head()` and `btron_locate_source()` *before* calling `btron_update_audio()`.

Other than waveform-playback interrupts, this is the only PC↔DSP interaction after `btron_init()`.

Because the Beachtron's internal DSP audio parameters can be updated only once every 46 msec (see discussion "Audio spatialization" in Chapter 1), more frequent calls to this routine may be ignored by the DSP spatialization process. However, if an update must be assured of getting made, the return value may be repeatedly tested until all Beachtrons are up-to-date:

```
while (btron_update_audio() < Ok) ;
```

6.3 Anhang C

6.3.1 Quellcode der beiden Programme

In diesem Abschnitt ist der Quellcode von beiden Programmen abgedruckt, die ich für die MAA-Untersuchung und der absoluten Untersuchung geschrieben habe.

Programm für die MAA-Untersuchung

```
#include <conio.h>                                      /* getch() && kbhit() */
#include <stdio.h>                                      /* printf() */
#include <dos.h>                                        /* delay() */
#include <math.h>                                       /* tan() */
#include <iomanip.h>
#include <iostream.h>
#include <time.h>
#include <stdlib.h>
#include <string.h>
#include "btron.h"

#define SourceID    0
#define Sources     1           /* number of sources */
#define WaveFile1    "fehzaj.WAV" /* filename */
#define WaveFile2    "1500hz.WAV"
#define WaveFile3    "7000hz.wav"
#define WaveFile4    "signal.WAV"
#define PanLimit    100.0
#define PI                      3.141592654
#define SOURCEDISTANCE          50.00000
#define RESOLUTION  1
#define EARDISTANCE INTERAURAL        /* default INTERAURAL = 6.0 inch, a ket ful tavolsaga  */
#define CM                      2.54                    /* 1 inch = 2.54 cm */

float  transformation(float,float); /* Funktion fuer die Koordinatentransformation*/



void main(void)
{
```

```
	float
ohrabstand,Aufloesungy,Aufloesungz,ywerte[121],zwerte[121],actualdegy,actualdegz,ypositiv[121],ynegativ[121],zpositiv[121],znegativ[121],amplification;
	char question1,question2,question3,question4,question5,question6,direction ;
	time_t first,second;
	int i,j,k,yp,yn,zp,zn,h;
	int yanzahl,zanzahl,origowertz,origowerty;
	FILE *referenzdatei;
	char * pers;
	char person[3];




	printf("Do you want to enter a value for the eardistance?\n");
	jump1: printf("type 'y' for 'yes' and 'n' for 'no':  ");
	//scanf("%c",&question1);
	cin  >> question1;
	if (question1!='y'&&question1!='n')
		{
		 printf("\nWrong Button! Please type again\n");
		 goto jump1;
		}
	if (question1=='y')
			{	marker6: printf("\nPlease enter a value between 15 cm - 50 cm for the eardistance:\n");
				scanf("%f",&ohrabstand);
				printf("%2.2f cm\n", ohrabstand);
				if(ohrabstand<15.0 || ohrabstand>50.0)
				  {
					printf("\nValue out of range. Please type again.\n");
					goto marker6;
				  }
				ohrabstand=ohrabstand/CM;
				printf("%2.2f inch\n", ohrabstand);
				float offsets[3]={0.0, ohrabstand, 0.0};
			 h=btron_model_head(offsets, NULL);
			 //printf("%i",h);
			 if (h<0)
				{
				 printf("\nError during modelling the headsize!\n");
				}
			}
	else
		{  printf("\nThe default value of 6.0 inch will be used for the eardistance!\n");
			float offsets[3]={0.0, INTERAURAL, 0.0};
```

```
                h=btron_model_head(offsets, NULL);
                //printf("%i",h);
                if (h<0)
                        {
                        printf("\nError during modelling the headsize!\n");
                        }
        };
printf("\nDo you want to enter a value for the resolution?\n");
jump2: printf("type 'y' for 'yes' and 'n' for 'no': ");
cin  >> question2;
/*printf("%c\n", question2);*/

if (question2!='y'&&question2!='n')
        {
         printf("\nWrong Button! Please type again\n");
         goto jump2;
        }

if (question2=='y')
                {       marker10: printf("Please enter a value for the resolution of the y-axis: ");
                        scanf("%f",&Aufloesungy);
                        printf("\nYou have chosen a Resolution of %2.2f degree\n", Aufloesungy);
                 if (Aufloesungy<1.0)
                                {
                                 printf("\nError! You must not type a value smaller than 1 degree!\n");
                                 goto marker10;
                                }
                        marker12: printf("Please enter a value for the resolution of the z-axis: ");
                        scanf("%f",&Aufloesungz);
                        printf("\nYou have chosen a Resolution of %2.2f degree\n", Aufloesungz);
                 if (Aufloesungz<1.0)
                                {
                                 printf("\nError! You must not type a value smaller than 1 degree!\n");
                                 goto marker12;
                                }

                }
else
        {       printf("\nThe default value for the resolution will be used!\n");
         };

printf("\nDo you want to enter a value for the amplification of the sound source?\n");
jump3: printf("type 'y' for 'yes' and 'n' for 'no': ");
cin  >> question5;
```

```
if (question5!='y'&&question5!='n')
        {
         printf("\nWrong Button! Please type again\n");
         goto jump3;
        }


if (question5=='y')
        {
          marker11: printf("\nPlease enter a value for the amplification: ");
                        scanf("%f",&amplification);
                        printf("\nYou have chosen an amplification of: %2.1f dB\n",amplification);
                        if (amplification<0.0)
                                {
                                 printf("\nError! You must not type a negativ value for the amplification!\n");
                                 goto marker11;
                                }
        }
else
        {
         printf("\nThe default value of 0 dB for the amplification will be used!\n");
         amplification=0;
        };

 printf("\nPlease type a personal ID consisting of 3 letters!\n");
 scanf("%3s",&person);
 printf("\n%s\n",person);
 sprintf(pers,"%s",person);
 printf("\n%s\n",pers);

if (btron_init(Sources,_VERBOSE_) < Ok)  return;

{
        wavFt *wave1  = btron_open_wave(WaveFile1, 4);
        if (wave1 == NULL)
                {
                 printf("\nA=%s failed to open.\n",WaveFile1);
                }
        wavFt *wave2 = btron_open_wave(WaveFile2, 4);
        if (wave2 == NULL)
                {
                 printf("\nB=%s failed to open.\n",WaveFile2);
                }
        wavFt *wave3  = btron_open_wave(WaveFile3, 4);
        if (wave3 == NULL)
```

```
                {
                 printf("\nC=%s failed to open.\n",WaveFile3);
                }
        wavFt *wave4  = btron_open_wave(WaveFile4, 4);
        if (wave4 == NULL)
                {
                 printf("\nD=%s failed to open.\n",WaveFile4);
                }
        marker5: printf("\nPlease select a source:A=white noise;B=1500 Hertz;C= 7000 Hertz;D= Signal.WAV.\n");
        question4=getch();

        if (question4=='a' ||question4=='A')
                {
                 printf("\nYou have chosen 'A=white noise' as your sound source!\n");
                 if (wave1!=NULL)
                        {
                                printf("\nWavefile loaded succesfully: %s \n",(*wave1).fname);
                                cputs("Max.length of the wav file (played only from the memory): 360 ms.");
                                printf("\nSampleRate: %5.0f Hz, Sample Size: %d bit, FrameSize: %d byte \nTotal Frames in file:
%ld",(*wave1).sampleRate, (*wave1).sampleSize*8, (*wave1).frameSize, (*wave1).numFrames);
                                printf("\nLength of signal selection: %ld, Beginning of signal selection: %ld \nNot yet loaded frames: %ld, %d channel.
\n",(*wave1).selFrames, (*wave1).startFrame, (*wave1).remFrames, (*wave1).numChannels);
                        }
                 strcat(pers,"-A.txt");
                 printf("\n%s\n",pers);
                 btron_amplfy_source(SourceID, GAIN_dB_ON+amplification);
                }
                if (question4=='b' ||question4=='B')
                {
                 printf("\nYou have chosen 'B=1500 Hertz' as your sound source!\n");

                 if (wave2!=NULL)
                        {
                                printf("\nWavefile loaded succesfully: %s \n",(*wave2).fname);
                                cputs("Max.length of the wav file (played only from the memory): 360 ms.");
                                printf("\nSampleRate: %5.0f Hz, Sample Size: %d bit, FrameSize: %d byte \nTotal Frames in file:
%ld",(*wave2).sampleRate, (*wave2).sampleSize*8, (*wave2).frameSize, (*wave2).numFrames);
                                printf("\nLength of signal selection: %ld, Beginning of signal selection: %ld \nNot yet loaded frames: %ld, %d channel.
\n",(*wave2).selFrames, (*wave2).startFrame, (*wave2).remFrames, (*wave2).numChannels);
                        }

                 strcat(pers,"-B.txt");
                 printf("\n%s\n",pers);
                 btron_amplfy_source(SourceID, GAIN_dB_ON+amplification+10);
                }
```

```
                        if (question4=='c' ||question4=='C')
                        {
                         printf("\nYou have chosen 'C=7000 Hertz' as your sound source!\n");

                         if (wave3!=NULL)
                                {
                                        printf("\nWavefile loaded succesfully: %s \n",(*wave3).fname);
                                        cputs("Max.length of the wav file (played only from the memory): 360 ms.");
                                        printf("\nSampleRate: %5.0f Hz, Sample Size: %d bit, FrameSize: %d byte \nTotal Frames in file:
%ld",(*wave3).sampleRate, (*wave3).sampleSize*8, (*wave3).frameSize, (*wave3).numFrames);
                                        printf("\nLength of signal selection: %ld, Beginning of signal selection: %ld \nNot yet loaded frames: %ld, %d channel.
\n",(*wave3).selFrames, (*wave3).startFrame, (*wave3).remFrames, (*wave3).numChannels);
                                }

                         strcat(pers,"-C.txt");
                         printf("\n%s\n",pers);
                         btron_amplfy_source(SourceID, GAIN_dB_ON+amplification+6);
                        }
                        if (question4=='d' ||question4=='D')
                        {
                         printf("\nYou have chosen 'D=Signal.WAV' as your sound source!\n");

                         if (wave4!=NULL)
                                {
                                        printf("\nWavefile loaded succesfully: %s \n",(*wave4).fname);
                                        cputs("Max.length of the wav file (played only from the memory): 360 ms.");
                                        printf("\nSampleRate: %5.0f Hz, Sample Size: %d bit, FrameSize: %d byte \nTotal Frames in file:
%ld",(*wave4).sampleRate, (*wave4).sampleSize*8, (*wave4).frameSize, (*wave4).numFrames);
                                        printf("\nLength of signal selection: %ld, Beginning of signal selection: %ld \nNot yet loaded frames: %ld, %d channel.
\n",(*wave4).selFrames, (*wave4).startFrame, (*wave4).remFrames, (*wave4).numChannels);
                                }

                         strcat(pers,"-D.txt");
                         printf("\n%s\n",pers);
                         btron_amplfy_source(SourceID, GAIN_dB_ON+amplification);
                        }
                        if (question4!='a'&&question4!='b'&&question4!='c'&&question4!='d'&&question4!='A'&&question4!='B'&&question4!='C'&&question4!='D')
                        {
                        printf("\nWrong Button!\n");
                        goto marker5;
                        }

                //btron_amplfy_source(SourceID, GAIN_dB_ON);
```

```
//btron_amplfy_source(1, GAIN_dB_ON+10);

/*y=transformation(SOURCEDISTANCE,15);
printf("\n%f\n", y);*/

if (question2!='y')
        {
                /*yanzahl=13;
                zanzahl=5;
                origowerty=6;
                origowertz=2;*/


                ywerte[0]= transformation(SOURCEDISTANCE,47.9);
                ywerte[1]= transformation(SOURCEDISTANCE,44);
                ywerte[2]= transformation(SOURCEDISTANCE,36.3);
                ywerte[3]= transformation(SOURCEDISTANCE,28.3);
                ywerte[4]= transformation(SOURCEDISTANCE,18.9);
                ywerte[5]= transformation(SOURCEDISTANCE,9.4);
                ywerte[6]= transformation(SOURCEDISTANCE,0);
                ywerte[7]= transformation(SOURCEDISTANCE,-7.6);
                ywerte[8]= transformation(SOURCEDISTANCE,-15.8);
                ywerte[9]= transformation(SOURCEDISTANCE,-24.9);
                ywerte[10]= transformation(SOURCEDISTANCE,-34);
                ywerte[11]= transformation(SOURCEDISTANCE,-42.5);
                ywerte[12]= transformation(SOURCEDISTANCE,-48.6);

                zwerte[0]=  transformation(SOURCEDISTANCE,33.3);
                zwerte[1]=  transformation(SOURCEDISTANCE,16.7);
                zwerte[2]=  transformation(SOURCEDISTANCE,0);
                zwerte[3]=  transformation(SOURCEDISTANCE,-16.0);
                zwerte[4]=  transformation(SOURCEDISTANCE,-31.6);

                float origin[6]={SOURCEDISTANCE,ywerte[6],zwerte[2],0.0,0.0,0.0};
                float location2[6]={SOURCEDISTANCE,ywerte[6],zwerte[2],0.0,0.0,0.0};
                k=0; j=0; yp=0;yn=0;zp=0;zn=0;
                if (k==0 && j==0)
                                        {
                                        marker4: printf("\nPlease use the 'u','d','l','r' keys to move the second source up,down,left or right. \n");
                                        direction=getch();
                                        if (direction=='u')
                                                {
                                                 zp=0;
                                                 printf("\nYou have moved the sound source up!\n");
```

```
                }
        if (direction=='d')
                {
                 zn=0;
                 printf("\nYou have moved the sound source down!\n");
                }
        if (direction=='r')
                {
                 yn=0;
                 printf("\nYou have moved the sound source to right!\n");
                }
        if (direction=='l')
                {
                 yp=0;
                 printf("\nYou have moved the sound source to left!\n");
                }
        if (direction!='u'&&direction!='d'&&direction!='l'&&direction!='r')
                {
                printf("\nWrong Button!\n");
                goto marker4;
                }

        }

printf("\nThe location of both sources is in origo.\n");
printf("The reference point is at: x=50 , y=0.0 , z=0.0 \n");
printf("Please press any key to start.\n");



if (getch()!=0)
        {
                goto marker3;
                repeat1:

                if (k==0 && j==0)
                        {
                        marker1: printf("\nPlease use the 'u','d','l','r' keys to move the second source up,down,left or right. \n");
                        direction=getch();
                        if (direction=='u')
                                {
                                 zp=0;
                                 printf("\nYou have moved the sound source up!\n");
                                }
                        if (direction=='d')
```

```
                {
                 zn=0;
                 printf("\nYou have moved the sound source down!\n");
                }
        if (direction=='r')
                {
                 yn=0;
                 printf("\nYou have moved the sound source to right!\n");
                }
        if (direction=='l')
                {
                 yp=0;
                 printf("\nYou have moved the sound source to left!\n");
                }
        if (direction!='u'&&direction!='d'&&direction!='l'&&direction!='r')
                {
                printf("\nWrong Button!\n");
                goto marker1;
                }
        }

marker3:        i=0;

switch(question4)
                {
                  case 'a': case 'A': btron_ctrl_wave(SourceID, wave1, WaveCTRL_STRT, NULL); break;
                  case 'b': case 'B': btron_ctrl_wave(SourceID, wave2, WaveCTRL_STRT, NULL); break;
                  case 'c': case 'C': btron_ctrl_wave(SourceID, wave3, WaveCTRL_STRT, NULL); break;
                  case 'd': case 'D': btron_ctrl_wave(SourceID, wave4, WaveCTRL_STRT, NULL); break;
                }

actualdegy= (atan((origin[AtrnY]/SOURCEDISTANCE)))*180/PI;
actualdegz= (atan((origin[AtrnZ]/SOURCEDISTANCE)))*180/PI;
printf("\n1st sound:Reference point: Y=%6.3f degree,Z=%6.3f degree",actualdegy,actualdegz);
btron_locate_source(SourceID, origin);

do {
        btron_update_audio();
        i=i+1;
        } while(i<32000);
switch(question4)
                {
                  case 'a': case 'A': btron_ctrl_wave(SourceID, wave1, WaveCTRL_STOP, NULL); break;
                  case 'b': case 'B': btron_ctrl_wave(SourceID, wave2, WaveCTRL_STOP, NULL); break;
                  case 'c': case 'C': btron_ctrl_wave(SourceID, wave3, WaveCTRL_STOP, NULL); break;
```

```
                    case 'd': case 'D': btron_ctrl_wave(SourceID, wave4, WaveCTRL_STOP, NULL); break;
                    }


                    /*first = time(NULL);
do {
          second = time(NULL);
                    } while(difftime(second,first)<=0.1);*/


switch(question4)
                    {
                      case 'a': case 'A': btron_ctrl_wave(SourceID, wave1, WaveCTRL_STRT, NULL); break;
                      case 'b': case 'B': btron_ctrl_wave(SourceID, wave2, WaveCTRL_STRT, NULL); break;
                      case 'c': case 'C': btron_ctrl_wave(SourceID, wave3, WaveCTRL_STRT, NULL); break;
                      case 'd': case 'D': btron_ctrl_wave(SourceID, wave4, WaveCTRL_STRT, NULL); break;
                    }
actualdegy= (atan((location2[AtrnY]/SOURCEDISTANCE)))*180/PI;
actualdegz= (atan((location2[AtrnZ]/SOURCEDISTANCE)))*180/PI;
printf("\n2nd sound:Moving point: Y=%6.3f degree ,Z=%6.3f degree",actualdegy,actualdegz);
btron_locate_source(SourceID, location2);
i=0;
do {
          btron_update_audio();
          i=i+1;
          } while(i<32000);
switch(question4)
                    {
                      case 'a': case 'A': btron_ctrl_wave(SourceID, wave1, WaveCTRL_STOP, NULL); break;
                      case 'b': case 'B': btron_ctrl_wave(SourceID, wave2, WaveCTRL_STOP, NULL); break;
                      case 'c': case 'C': btron_ctrl_wave(SourceID, wave3, WaveCTRL_STOP, NULL); break;
                      case 'd': case 'D': btron_ctrl_wave(SourceID, wave4, WaveCTRL_STOP, NULL); break;
                    }

question3='n';
first = time(NULL);

do {

          if (kbhit() && getch()==32)
                    {
                    //printf("\nCould you perceice any difference in the location of the two sources?\n");
                    //printf("\nType 'y' for 'yes' and 'n' for 'no' or 'r' for 'repeat': \n");
                    question3='y';
                    }
```

```
                                                        second = time(NULL);
                                                        } while(difftime(second,first)<=1.0);

                                                        if (( question3=='y' || question3=='n'))
                                                                                                {
                                                                                                        if (question3=='y')
                                                                                                                {
                                                                                                                        actualdegy=
(atan((location2[AtrnY]/SOURCEDISTANCE)))*180/PI;
                                                                                                                        actualdegz=
(atan((location2[AtrnZ]/SOURCEDISTANCE)))*180/PI;
                                                                                                                        printf("\nThe new
reference point is in: (y=%2.2f degree, z=%2.2f degree) \n",actualdegy,actualdegz);
                origin[AtrnY]=location2[AtrnY];
                origin[AtrnZ]=location2[AtrnZ];

                                                                                                                        if (direction=='r')
                                                                                                                                {
                                                                                                                                  ynegativ[yn]=
actualdegy;
                                                                                                                                  yn=yn++;
                                                                                                                                }

                                                                                                                        if (direction=='l')
                                                                                                                                {
                                                                                                                                  ypositiv[yp]=
actualdegy;
                                                                                                                                  yp=yp++;

                                                                                                                                }

                                                                                                                        if (direction=='u')
                                                                                                                                {
                                                                                                                                  zpositiv[zp]=
actualdegz;
                                                                                                                                  zp=zp++;
                                                                                                                                }

                                                                                                                        if (direction=='d')
                                                                                                                                {
                                                                                                                                  znegativ[zn]=
actualdegz;
                                                                                                                                  zn=zn++;
```

```
                                        }
                                if (k==0 && j==0)
                                        {
                                         goto marker3;
                                        }
                                  else
                                        {
                                         goto repeat1;
                                        }
                        }
else
                        {
                                actualdegy=

                                actualdegz=

                                printf("\nThe reference

                        };

/*if (direction=='u' ||direction=='d'

                        {
                         goto marker1;
                        }*/


/*if (k==0 && j==0)
                        {
                                printf("\nPlease use the

                        }
else
                        {
                          if (k==0)
                                        {


                                        }
                           else
                                        {


                                        };
                           };*/
```

(atan((origin[AtrnY]/SOURCEDISTANCE)))*180/PI;

(atan((origin[AtrnZ]/SOURCEDISTANCE)))*180/PI;

point is still in: (y=%2.2f degree, z=%2.2f degree) \n",actualdegy,actualdegz);

||direction=='l' ||direction=='r')

'u','d','l','r' keys to move the second source up,down,left or right. \n");

printf("\nPlease use the 'l','r' keys to move the second source to left or right. \n");

printf("\nPlease use the 'u','d' keys to move the second source up or down. \n");

```
                                                                    //repeat3: direction=getch();
                                                                                    //printf("%c",direction);


                                                                    switch (direction)
                                                                                                    {
                                                                                                    case 'u': if(j==0)
{
        if (k==-2)
                {
                        printf("\nYou can not move the source further up.\n");
                        label1: printf("\nDo you want to start in another direction?\n");
                        printf("\nPlease type 'y' for 'yes' or 'n' for 'no'.\n");
                        question6= getch();
                        if (question6!='y'&&question6!='n')
                                {
                                 printf("\nWrong Button! Please type again!\n");
                                 goto label1;
                                }
                        if (question6=='y')
                                {
                                  k=0; j=0;
                                  origin[1]=ywerte[6];
                                  origin[2]=zwerte[2];
                                  location2[1]=ywerte[6];
```

```
                                        location2[2]=zwerte[2];
                                        goto repeat1;
                                      }
                                else
                                      {
                                        goto exit;
                                      };
                        };
        k=k--;  location2[AtrnZ]= zwerte[2+k];
}
break;
                                                                                                    case 'd': if(j==0)
{
        if (k==+2)
                        {
                          printf("\nYou can not move the source further down.\n");
                          label2: printf("\nDo you want to start in another direction?\n");
                          printf("\nPlease type 'y' for 'yes' or 'n' for 'no'.\n");
                        question6= getch();
                          if (question6!='y'&&question6!='n')
                                      {
                                       printf("\nWrong Button! Please type again!\n");
                                       goto label2;
```

```
                                        }
                                if (getch()=='y')
                                        {
                                          k=0; j=0;
                                          origin[1]=ywerte[6];
                                          origin[2]=zwerte[2];
                                          location2[1]=ywerte[6];
                                          location2[2]=zwerte[2];
                                          goto repeat1;
                                        }
                                 else
                                        {
                                          goto exit;
                                        };
                        };
        k=k++;  location2[AtrnZ]= zwerte[2+k];
}
break;                                                                                                  case 'l': if(k==0)
{
        if (j==-6)
                        {
                                printf("\nYou can not move the source further left.\n");
```

```
label3: printf("\nDo you want to start in another direction?\n");
printf("\nPlease type 'y' for 'yes' or 'n' for 'no'.\n");
question6= getch();
if (question6!='y'&&question6!='n')
        {
         printf("\nWrong Button! Please type again!\n");
         goto label3;
        }
if (getch()=='y')
        {
          k=0; j=0;
          origin[1]=ywerte[6];
          origin[2]=zwerte[2];
          location2[1]=ywerte[6];
          location2[2]=zwerte[2];
          goto repeat1;
        }
else
        {
          goto exit;
        };
};
```

```
                j=j--;  location2[AtrnY]= ywerte[6+j];
        }
break;
                                                                                                case 'r': if(k==0)
        {
                if (j==+6)
                        {
                          printf("\nYou can not move the source further right.\n");
                          label4: printf("\nDo you want to start in another direction?\n");
                          printf("\nPlease type 'y' for 'yes' or 'n' for 'no'.\n");
                        question6= getch();
                          if (question6!='y'&&question6!='n')
                                {
                                 printf("\nWrong Button! Please type again!\n");
                                 goto label4;
                                }
                          if (getch()=='y')
                                {
                                  k=0; j=0;
                                  origin[1]=ywerte[6];
                                  origin[2]=zwerte[2];
                                  location2[1]=ywerte[6];
                                  location2[2]=zwerte[2];
```

```
                                        goto repeat1;
                                        }
                                else
                                        {
                                        goto exit;
                                        };
                        };
                j=j++;  location2[AtrnY]= ywerte[6+j];
        }
break;
                                                                                                        }
                                                                                                        goto repeat1;
                                                                                }
                                }
        }
else
        {
          float a,c;
          int b,d;

          c=Aufloesungy*100; // um die Genauigkeit bei Kommazahlen zu erhõhen
          d=c;
          a=6000/d;         // um die Genauigkeit bei Kommazahlen zu erhõhen
          b=a;

          yanzahl=b*2+1;
          //zanzahl=yanzahl;
          origowerty=(yanzahl-1)/2;
          //origowerty=origowertz;

          c=Aufloesungz*100; // um die Genauigkeit bei Kommazahlen zu erhõhen
```

```
d=c;
a=6000/d;        // um die Genauigkeit bei Kommazahlen zu erhõhen
b=a;

zanzahl=b*2+1;
origowertz=(zanzahl-1)/2;

printf("\ny=%i,z=%i,oz=%i,oy=%i\n",yanzahl,zanzahl,origowertz,origowerty);

for (i=0; i<origowerty; i++)
        {
                ywerte[i]= transformation(SOURCEDISTANCE,(((origowerty-i)*Aufloesungy)));
                //zwerte[i]= transformation(SOURCEDISTANCE,(((origowerty-i)*Aufloesung)));
                printf("\ny=%f",ywerte[i]);

        };
        for (i=origowerty; i<yanzahl; i++)
        {
                ywerte[i]= transformation(SOURCEDISTANCE,( (origowerty-i)*Aufloesungy));
                //zwerte[i]= transformation(SOURCEDISTANCE,( (origowerty-i)*Aufloesung));
                printf("\ny=%f",ywerte[i]);
        };



        for (i=0; i<origowertz; i++)
        {
                zwerte[i]= transformation(SOURCEDISTANCE,(((origowertz-i)*Aufloesungz)));
                //zwerte[i]= transformation(SOURCEDISTANCE,(((origowerty-i)*Aufloesung)));
                printf("\nz=%f",zwerte[i]);

        };
        for (i=origowertz; i<zanzahl; i++)
        {
                zwerte[i]= transformation(SOURCEDISTANCE,( (origowertz-i)*Aufloesungz));
                //zwerte[i]= transformation(SOURCEDISTANCE,( (origowerty-i)*Aufloesung));
                printf("\nz=%f",zwerte[i]);
        };

        printf("\ny=%i,z=%i,oz=%i,oy=%i\n",yanzahl,zanzahl,origowertz,origowerty);
        printf("\n" );
        printf("\ny=%f,z=%f\n",ywerte[origowerty],zwerte[origowertz]);

        float location3[6]={SOURCEDISTANCE,ywerte[origowerty],zwerte[origowertz],0.0,0.0,0.0};
        float location4[6]={SOURCEDISTANCE,ywerte[origowerty],zwerte[origowertz],0.0,0.0,0.0};
```

```
k=0; j=0; yp=0;yn=0;zp=0;zn=0;
                    if (k==0 && j==0)
                                {
                                marker7: printf("\nPlease use the 'u','d','l','r' keys to move the second source up,down,left or right. \n");
                                direction=getch();
                                if (direction=='u')
                                        {
                                         zp=0;
                                         printf("\nYou have moved the sound source up!\n");
                                        }
                                if (direction=='d')
                                        {
                                         zn=0;
                                         printf("\nYou have moved the sound source down!\n");
                                        }
                                if (direction=='r')
                                        {
                                         yn=0;
                                         printf("\nYou have moved the sound source to right!\n");
                                        }
                                if (direction=='l')
                                        {
                                         yp=0;
                                         printf("\nYou have moved the sound source to left!\n");
                                        }
                                if (direction!='u'&&direction!='d'&&direction!='l'&&direction!='r')
                                        {
                                        printf("\nWrong Button!\n");
                                        goto marker7;
                                        }

                                }

                    printf("\nThe location of both sources is in origo.\n");
                    printf("The reference point is at: x=50 , y=0.0 , z=0.0 \n");
                    printf("Please press any key to start.\n");

                    if (getch()!=0)
                            {
                                        goto marker8;
                            repeat4:

                            if (k==0 && j==0)
                                        {
```

```
marker9: printf("\nPlease use the 'u','d','l','r' keys to move the second source up,down,left or right. \n");
direction=getch();
if (direction=='u')
        {
         zp=0;
         printf("\nYou have moved the sound source up!\n");
        }
if (direction=='d')
        {
         zn=0;
         printf("\nYou have moved the sound source down!\n");
        }
if (direction=='r')
        {
         yn=0;
         printf("\nYou have moved the sound source to right!\n");
        }
if (direction=='l')
        {
         yp=0;
         printf("\nYou have moved the sound source to left!\n");
        }
if (direction!='u'&&direction!='d'&&direction!='l'&&direction!='r')
        {
        printf("\nWrong Button!\n");
        goto marker9;
        }
}

marker8:        i=0;


switch(question4)
                {
                  case 'a': case 'A': btron_ctrl_wave(SourceID, wave1, WaveCTRL_STRT, NULL); break;
                  case 'b': case 'B': btron_ctrl_wave(SourceID, wave2, WaveCTRL_STRT, NULL); break;
                  case 'c': case 'C': btron_ctrl_wave(SourceID, wave3, WaveCTRL_STRT, NULL); break;
                  case 'd': case 'D': btron_ctrl_wave(SourceID, wave4, WaveCTRL_STRT, NULL); break;
                }

actualdegy= (atan((location3[AtrnY]/SOURCEDISTANCE)))*180/PI;
actualdegz= (atan((location3[AtrnZ]/SOURCEDISTANCE)))*180/PI;
printf("\n1st sound:Reference point: Y=%6.3f degree,Z=%6.3f degree",actualdegy,actualdegz);
btron_locate_source(SourceID, location3);
do {
```

```
            btron_update_audio();
            i=i+1;
            } while(i<32000);
switch(question4)
                        {
                          case 'a': case 'A': btron_ctrl_wave(SourceID, wave1, WaveCTRL_STOP, NULL); break;
                          case 'b': case 'B': btron_ctrl_wave(SourceID, wave2, WaveCTRL_STOP, NULL); break;
                          case 'c': case 'C': btron_ctrl_wave(SourceID, wave3, WaveCTRL_STOP, NULL); break;
                          case 'd': case 'D': btron_ctrl_wave(SourceID, wave4, WaveCTRL_STOP, NULL); break;

                        }



/*first = time(NULL);
do {
              second = time(NULL);
                        } while(difftime(second,first)<=0.1);*/



switch(question4)
                        {
                          case 'a': case 'A': btron_ctrl_wave(SourceID, wave1, WaveCTRL_STRT, NULL); break;
                          case 'b': case 'B': btron_ctrl_wave(SourceID, wave2, WaveCTRL_STRT, NULL); break;
                          case 'c': case 'C': btron_ctrl_wave(SourceID, wave3, WaveCTRL_STRT, NULL); break;
                          case 'd': case 'D': btron_ctrl_wave(SourceID, wave4, WaveCTRL_STRT, NULL); break;
                        }
actualdegy= (atan((location4[AtrnY]/SOURCEDISTANCE)))*180/PI;
actualdegz= (atan((location4[AtrnZ]/SOURCEDISTANCE)))*180/PI;
printf("\n2nd sound:Moving point: Y=%6.3f degree ,Z=%6.3f degree",actualdegy,actualdegz);
btron_locate_source(SourceID, location4);
i=0;
do {
            btron_update_audio();
            i=i+1;
            } while(i<32000);
switch(question4)
                        {
                          case 'a': case 'A': btron_ctrl_wave(SourceID, wave1, WaveCTRL_STOP, NULL); break;
                          case 'b': case 'B': btron_ctrl_wave(SourceID, wave2, WaveCTRL_STOP, NULL); break;
                          case 'c': case 'C': btron_ctrl_wave(SourceID, wave3, WaveCTRL_STOP, NULL); break;
                          case 'd': case 'D': btron_ctrl_wave(SourceID, wave4, WaveCTRL_STOP, NULL); break;
```

```
                                        }

                                question3='n';
                                first = time(NULL);

                                do {

                                        if (kbhit() && getch()==32)
                                                {
                                                //printf("\nCould you perceice any difference in the location of the two sources?\n");
                                                //printf("\nType 'y' for 'yes' and 'n' for 'no' or 'r' for 'repeat': \n");
                                                question3='y';
                                                }
                                                second = time(NULL);
                                                } while(difftime(second,first)<=1.0);

                                                if (( question3=='y' || question3=='n'))
                                                                                                {
                                                                                                                if (question3=='y')
                                                                                                                        {
                                                                                                                                actualdegy=
(atan((location4[AtrnY]/SOURCEDISTANCE)))*180/PI;
                                                                                                                                actualdegz=
(atan((location4[AtrnZ]/SOURCEDISTANCE)))*180/PI;
                                                                                                                                printf("\nThe new
reference point is in: (y=%2.2f degree, z=%2.2f degree) \n",actualdegy,actualdegz);
        location3[AtrnY]=location4[AtrnY];
        location3[AtrnZ]=location4[AtrnZ];

                                                                                                                                if (direction=='r')
                                                                                                                                        {
                                                                                                                                          ynegativ[yn]=
actualdegy;
                                                                                                                                          yn=yn++;
                                                                                                                                        }

                                                                                                                                if (direction=='l')
                                                                                                                                        {
                                                                                                                                          ypositiv[yp]=
actualdegy;
                                                                                                                                          yp=yp++;

                                                                                                                                        }
```

```
actualdegz;



actualdegz;

(atan((location3[AtrnY]/SOURCEDISTANCE)))*180/PI;

(atan((location3[AtrnZ]/SOURCEDISTANCE)))*180/PI;

point is still in: (y=%2.2f degree, z=%2.2f degree) \n",actualdegy,actualdegz);

||direction=='l' ||direction=='r')
```

```
                if (direction=='u')
                        {
                          zpositiv[zp]=

                          zp=zp++;
                        }

                if (direction=='d')
                        {
                          znegativ[zn]=

                          zn=zn++;
                        }
                 if (k==0 && j==0)
                        {
                         goto marker8;
                        }
                  else
                        {
                         goto repeat4;
                        }


        }
else
        {
                actualdegy=

                actualdegz=

                printf("\nThe reference

        };

/*if (direction=='u' ||direction=='d'

        {
         goto marker1;
        }*/


/*if (k==0 && j==0)
        {
```

```
                                                                                    printf("\nPlease use the
'u','d','l','r' keys to move the second source up,down,left or right. \n");
                                                                                                }
                                                                                    else
                                                                                                {
                                                                                                 if (k==0)
                                                                                                                {

        printf("\nPlease use the 'l','r' keys to move the second source to left or right. \n");
                                                                                                                }
                                                                                                 else
                                                                                                                {

        printf("\nPlease use the 'u','d'  keys to move the second source up or down.  \n");
                                                                                                                };
                                                                                                 };*/

                                                                                  //repeat3: direction=getch();
                                                                                                //printf("%c",direction);


                                                                                  switch (direction)
                                                                                                                {
                                                                                                case 'u': if(j==0)

                {

                        if (k==-(origowertz))

                                {

                                  printf("\nYou can not move the source further up.\n");

                                  label5: printf("\nDo you want to start in another direction?\n");

                                  printf("\nPlease type 'y' for 'yes' or 'n' for 'no'.\n");
                                 question6= getch();

                                  if (question6!='y'&&question6!='n')

                                          {

                                           printf("\nWrong Button! Please type again!\n");

                                           goto label5;
```

```
                        }
                    if (getch()=='y')
                        {
                          k=0; j=0;
                          location3[1]=ywerte[origowerty];
                          location3[2]=zwerte[origowertz];
                          location4[1]=ywerte[origowerty];
                          location4[2]=zwerte[origowertz];
                          goto repeat4;
                        }
                    else
                        {
                          goto exit;
                        };
                  };
        k=k--;  location4[AtrnZ]= zwerte[origowertz+k];
}
break;
                                                                                                    case 'd': if(j==0)
{
        if (k==+(origowertz))
                {
                    printf("\nYou can not move the source further down.\n");
```

```
label6: printf("\nDo you want to start in another direction?\n");

printf("\nPlease type 'y' for 'yes' or 'n' for 'no'.\n");
question6= getch();

if (question6!='y'&&question6!='n')

        {

         printf("\nWrong Button! Please type again!\n");

         goto label6;

        }

if (getch()=='y')

        {

          k=0; j=0;

          location3[1]=ywerte[origowerty];

          location3[2]=zwerte[origowertz];

          location4[1]=ywerte[origowerty];

          location4[2]=zwerte[origowertz];

          goto repeat4;

        }

 else

        {

          goto exit;

        };

    };

k=k++;  location4[AtrnZ]= zwerte[origowertz+k];
```

```
}
break;
                                                                                    case 'l': if(k==0)
{
        if (j==-(origowerty))
                {
                  printf("\nYou can not move the source further left.\n");
                  label7: printf("\nDo you want to start in another direction?\n");
                  printf("\nPlease type 'y' for 'yes' or 'n' for 'no'.\n");
                 question6= getch();
                  if (question6!='y'&&question6!='n')
                        {
                         printf("\nWrong Button! Please type again!\n");
                         goto label7;
                        }
                  if (getch()=='y')
                        {
                          k=0; j=0;
                          location3[1]=ywerte[origowerty];
                          location3[2]=zwerte[origowertz];
                          location4[1]=ywerte[origowerty];
                          location4[2]=zwerte[origowertz];
                          goto repeat4;
```

```
                                        }
                                else
                                        {
                                          goto exit;
                                        };
                        };
                j=j--;  location4[AtrnY]= ywerte[origowerty+j];
        }
break;                                                                                  case 'r': if(k==0)
        {
                if (j==+origowerty)
                        {
                                printf("\nYou can not move the source further right.\n");
                                label8: printf("\nDo you want to start in another direction?\n");
                                printf("\nPlease type 'y' for 'yes' or 'n' for 'no'.\n");
                                question6= getch();
                                if (question6!='y'&&question6!='n')
                                        {
                                         printf("\nWrong Button! Please type again!\n");
                                         goto label8;
                                        }
                                if (getch()=='y')
```

```
                                        {
                                          k=0; j=0;
                                          location3[1]=ywerte[origowerty];
                                          location3[2]=zwerte[origowertz];
                                          location4[1]=ywerte[origowerty];
                                          location4[2]=zwerte[origowertz];
                                          goto repeat4;
                                        }
                                else
                                        {
                                          goto exit;
                                        };
                        };
                j=j++;  location4[AtrnY]= ywerte[origowerty+j];
        }
break;
                                                                                        } // end of switch
                                                                                        goto repeat4;
                                                                } // end of if (question3=='y' or question3=='n')
```

```
                                        } // end of if (getch!=0)



                                };  // end of second if_else (question2== 'y' or question2== 'n' )
exit:

        printf("\n%i,%i,%i,%i\n",yp,yn,zp,zn);
        char * pfad;
        char  * satz;
        pfad="c:\\Gyor2004\\";
        strcat(pfad,pers);
        printf("\nPfad: %s\n",pfad);
        referenzdatei = fopen(pfad, "w");
        if (referenzdatei==NULL)
                {
                        printf("Fehler beim oeffnen der Datei!\n");
                }

        if (question2=='y')
                {
                        sprintf(satz,"Resolution of y-axis=%2.1f degree\n",Aufloesungy);
                        fprintf(referenzdatei,"%s\n",satz);
                        sprintf(satz,"Resolution of z-axis=%2.1f degree\n",Aufloesungz);
                        fprintf(referenzdatei,"%s\n",satz);

                }
               else
                {
                         satz="Resolution= default\n";
                         fprintf(referenzdatei,"%s\n",satz);

                };

        if (question5=='y')
                {
                        sprintf(satz,"Amplification=%2.1f dB\n",amplification);
                        fprintf(referenzdatei,"%s\n",satz);
                }
               else
                {
                         satz="Amplification= default\n";
                         fprintf(referenzdatei,"%s\n",satz);
```

```
                };

        satz="Reference points left to origo:\n";
        fprintf(referenzdatei,"%s\n",satz);
        for (i=0; i<yp; i++)
                        {
                         fprintf(referenzdatei,"Y=%2.2f degree\n",ypositiv[i]);
                        }
        satz="\nReference points right to origo:\n";
        fprintf(referenzdatei,"%s\n",satz);
        for (i=0; i<yn; i++)
                        {
                         fprintf(referenzdatei,"          Y=%2.2f degree\n",ynegativ[i]);
                        }
        satz="\nReference points upwards to origo:\n";
        fprintf(referenzdatei,"%s\n",satz);
        for (i=0; i<zp; i++)
                        {
                         fprintf(referenzdatei,"Z=%2.2f degree\n",zpositiv[i]);
                        }
        satz="\nReference points downwards to origo:\n";
        fprintf(referenzdatei,"%s\n",satz);
        for (i=0; i<zn; i++)
                        {
                         fprintf(referenzdatei,"          Z=%2.2f degree\n",znegativ[i]);
                        }

        fclose(referenzdatei);

        btron_close_wave(wave1);
        btron_close_wave(wave2);
        btron_close_wave(wave3);
        btron_close_wave(wave4);
        btron_close();


      } // end of first if (btron_init)


} // end of main

float  transformation(float quellenabstand, float degree)
```

```
{
        float xy;
        xy = quellenabstand*(tan(degree*(PI/180)));
        return xy;
}
```

Programm für die absolute Untersuchung

```
#include <conio.h>                                      /* getch() && kbhit() */
#include <stdio.h>                                      /* printf() */
#include <dos.h>                                        /* delay() */
#include <math.h>                                       /* tan() */
#include <iomanip.h>
#include <iostream.h>
#include <time.h>
#include <stdlib.h>
#include <string.h>
#include "btron.h"

#define SourceID    0
#define Sources     2           /* number of sources */
#define WaveFile1    "fehzaj.WAV" /* filename */
#define WaveFile2    "1500hz.WAV"
#define WaveFile3    "7000hz.wav"
#define WaveFile4    "signal.WAV"
#define PanLimit    100.0
#define PI                      3.141592654
#define SOURCEDISTANCE          50.00000
#define RESOLUTION  1
#define EARDISTANCE INTERAURAL        /* default INTERAURAL = 6.0 inch, a ket ful tavolsaga  */
#define CM                      2.54                    /* 1 inch = 2.54 cm */

float  transformation(float,float); /* Funktion fuer die Koordinatentransformation*/
int    vectorcoordinates(char[2], int, int);

struct vektortyp {
                                                char v;
                                                char h;
```

```
                                                    };


void main(void)
{

float ohrabstand,ywerte[7],zwerte[5],actualdegy,actualdegz,amplification;
char question1,question2,question3,question4,question5,newround;
//time_t first,second;
int i,j,k,ver,a,horizontal,vertical,key,res,index,repeat,r,s,round,roundcounter,m,n,chkrnd,pause;
FILE *referenzdatei;
long u;
char  *pers;
char window[2],window1[2];
char letter1[5]={'a','b','c','d','e'};
char letter2[7]={'1','2','3','4','5','6','7'};

float location[6]={SOURCEDISTANCE,0,0,0.0,0.0,0.0};
char person[3];
char * pfad;
char  * satz,* satz1;
vektortyp vektor1[36];
vektortyp vektor2[35];
time_t first,second;
time_t t;
int matrix[35][35];

for (m=0;m<35;m++)
            {
                        for (n=0;n<35;n++)
                                    {
                                     matrix[m][n]=0;
                                    }
             }
printf("Do you want to enter a value for the eardistance?\n");
jump1: printf("type 'y' for 'yes' and 'n' for 'no':  ");
//scanf("%c",&question1);
cin  >> question1;
if (question1!='y'&&question1!='n')
            {
            printf("\nWrong Button! Please type again\n");
            goto jump1;
            }
if (question1=='y')
            {
```

```
        marker1: printf("\nPlease enter a value between 15 cm - 50 cm for the eardistance:\n");
        scanf("%f",&ohrabstand);
        printf("%2.2f cm\n", ohrabstand);
        if(ohrabstand<15.0 || ohrabstand>50.0)
          {
          printf("\nValue out of range. Please type again.\n");
          goto marker1;
          }
        ohrabstand=ohrabstand/CM;
        printf("%2.2f inch\n", ohrabstand);
        float offsets[3]={0.0, ohrabstand, 0.0};
        a=btron_model_head(offsets, NULL);
        //printf("%i",h);
        if (a<0)
                {
                printf("\nError during modelling the headsize!\n");
                }
        }
        else
                {
                printf("\nThe default value of 6.0 inch will be used for the eardistance!\n");
                float offsets[3]={0.0, INTERAURAL, 0.0};
                a=btron_model_head(offsets, NULL);
                //printf("%i",h);
                if (a<0)
                        {
                        printf("\nError during modelling the headsize!\n");
                        }
                };

marker2 :printf("\nPlease enter a value for the vertical resolution(value must be between 1-5):\n");
scanf("%i",&vertical);
printf("\nYou have chosen the following vertical resolution: %i\n",vertical);

if(vertical<1 || vertical>5)
  {
        printf("\nValue out of range. Please type again.\n");
        goto marker2;
  }

marker3 :printf("\nPlease enter a value for the horizontal resolution(value must be between 3-7):\n");
scanf("%i",&horizontal);
printf("\nYou have chosen the following horizontal resolution: %i\n",horizontal);

if(horizontal<3 || horizontal>7)
```

```
 {
        printf("\nValue out of range. Please type again.\n");
        goto marker3;
 }

printf("\nDo you want to enter a value for the amplification of the sound source?\n");
jump2: printf("type 'y' for 'yes' and 'n' for 'no': ");
cin  >> question2;

if (question2!='y'&&question2!='n')
        {
        printf("\nWrong Button! Please type again\n");
        goto jump2;
        }


if (question2=='y')
        {
         marker4: printf("\nPlease enter a value for the amplification: ");
         scanf("%f",&amplification);
         printf("\nYou have chosen an amplification of: %2.1f dB\n",amplification);
         if (amplification<0.0)
                {
                printf("\nError! You must not type a negativ value for the amplification!\n");
                goto marker4;
                }
        }
        else
        {
         printf("\nThe default value of 0 dB for the amplification will be used!\n");
         amplification=0;
        };

        /*printf("\nPlease type a personal ID consisting of 3 letters!\n");
        scanf("%3s",&person);
        printf("\n%s\n",person);
        sprintf(pers,"%s",person);
        printf("\n%s\n",pers);*/


if (btron_init(Sources,_VERBOSE_) < Ok)  return;

        {
          wavFt *wave1  = btron_open_wave(WaveFile1, 4);
          if (wave1 == NULL)
```

```
		{
		printf("\nA=%s failed to open.\n",WaveFile1);
		}
	wavFt *wave2 = btron_open_wave(WaveFile2, 4);
	if (wave2 == NULL)
		{
		printf("\nB=%s failed to open.\n",WaveFile2);
		}
	wavFt *wave3  = btron_open_wave(WaveFile3, 4);
	if (wave3 == NULL)
		{
		printf("\nC=%s failed to open.\n",WaveFile3);
		}
	wavFt *wave4  = btron_open_wave(WaveFile4, 4);
	if (wave4 == NULL)
		{
		printf("\nD=%s failed to open.\n",WaveFile4);
		}

	question5='a';
	if(vertical==2 || vertical==3)
		{
			if(vertical==2)
			 {
				printf("\nDo you want 2 different sound sources (HPF noise for up and LPF noise for down) for the vertical resolution?\n");
				jump4: printf("type 'y' for 'yes' and 'n' for 'no':  ");
				cin  >> question5;
				if (question5!='y'&&question5!='n')
					{
					 printf("\nWrong Button! Please type again\n");
					 goto jump4;
					}
			 }
			if(vertical==3)
			 {
				printf("\nDo you want a 3 different sound sources (HPF noise for up and LPF noise for down and white noise for origo) for the
vertical resolution?\n");
				jump5: printf("type 'y' for 'yes' and 'n' for 'no':  ");
				cin  >> question5;
				if (question5!='y'&&question5!='n')
					{
					 printf("\nWrong Button! Please type again\n");
					 goto jump5;
					}
			 }
```

```
                }

        if (question5!='y')
                {
                 marker5: printf("\nPlease select a source:A=white noise;B=1500 Hertz;C= 7000 Hertz;D= Signal.WAV.\n");
                        question3=getch();
                }


        if (question3=='a' ||question3=='A')
                {
                printf("\nYou have chosen 'A=white noise' as your sound source!\n");
                if (wave1!=NULL)
                        {
                        printf("\nWavefile loaded succesfully: %s \n",(*wave1).fname);
                        cputs("Max.length of the wav file (played only from the memory): 360 ms.");
                        printf("\nSampleRate: %5.0f Hz, Sample Size: %d bit, FrameSize: %d byte \nTotal Frames in file: %ld",(*wave1).sampleRate,
(*wave1).sampleSize*8, (*wave1).frameSize, (*wave1).numFrames);
                        printf("\nLength of signal selection: %ld, Beginning of signal selection: %ld \nNot yet loaded frames: %ld, %d channel.
\n",(*wave1).selFrames, (*wave1).startFrame, (*wave1).remFrames, (*wave1).numChannels);
                        }
                //strcat(pers,"-A.txt");
                //printf("\n%s\n",pers);
                btron_amplfy_source(SourceID, GAIN_dB_ON+amplification);
                }
        if (question3=='b' ||question3=='B')
                {
                printf("\nYou have chosen 'B=1500 Hertz' as your sound source!\n");
                if (wave2!=NULL)
                        {
                        printf("\nWavefile loaded succesfully: %s \n",(*wave2).fname);
                        cputs("Max.length of the wav file (played only from the memory): 360 ms.");
                        printf("\nSampleRate: %5.0f Hz, Sample Size: %d bit, FrameSize: %d byte \nTotal Frames in file: %ld",(*wave2).sampleRate,
(*wave2).sampleSize*8, (*wave2).frameSize, (*wave2).numFrames);
                        printf("\nLength of signal selection: %ld, Beginning of signal selection: %ld \nNot yet loaded frames: %ld, %d channel.
\n",(*wave2).selFrames, (*wave2).startFrame, (*wave2).remFrames, (*wave2).numChannels);
                        }
                        //strcat(pers,"-B.txt");
                        btron_amplfy_source(SourceID, GAIN_dB_ON+amplification+10);
                }
        if (question3=='c' ||question3=='C')
                {
                printf("\nYou have chosen 'C=7000 Hertz' as your sound source!\n");
                if (wave3!=NULL)
                        {
```

```
                                printf("\nWavefile loaded succesfully: %s \n",(*wave3).fname);
                                cputs("Max.length of the wav file (played only from the memory): 360 ms.");
                                printf("\nSampleRate: %5.0f Hz, Sample Size: %d bit, FrameSize: %d byte \nTotal Frames in file: %ld",(*wave3).sampleRate,
(*wave3).sampleSize*8, (*wave3).frameSize, (*wave3).numFrames);
                                printf("\nLength of signal selection: %ld, Beginning of signal selection: %ld \nNot yet loaded frames: %ld, %d channel.
\n",(*wave3).selFrames, (*wave3).startFrame, (*wave3).remFrames, (*wave3).numChannels);
                                }
                                //strcat(pers,"-C.txt");
                                btron_amplfy_source(SourceID, GAIN_dB_ON+amplification+6);
                        }
                if (question3=='d' ||question3=='D')
                        {
                        printf("\nYou have chosen 'D=Signal.WAV' as your sound source!\n");
                        if (wave4!=NULL)
                                {
                                printf("\nWavefile loaded succesfully: %s \n",(*wave4).fname);
                                cputs("Max.length of the wav file (played only from the memory): 360 ms.");
                                printf("\nSampleRate: %5.0f Hz, Sample Size: %d bit, FrameSize: %d byte \nTotal Frames in file: %ld",(*wave4).sampleRate,
(*wave4).sampleSize*8, (*wave4).frameSize, (*wave4).numFrames);
                                printf("\nLength of signal selection: %ld, Beginning of signal selection: %ld \nNot yet loaded frames: %ld, %d channel.
\n",(*wave4).selFrames, (*wave4).startFrame, (*wave4).remFrames, (*wave4).numChannels);
                                }
                        //strcat(pers,"-D.txt");
                        btron_amplfy_source(SourceID, GAIN_dB_ON+amplification);
                        }
                if (question3!='a'&&question3!='b'&&question3!='c'&&question3!='d'&&question3!='A'&&question3!='B'&&question3!='C'&&question3!='D'&&question5!='y')
                        {
                        printf("\nWrong Button!\n");
                        goto marker5;
                        }


                marker8: printf("\nPlease enter a value between 0 msec-7000 msec for the break between the sound   beeps:\n");
                scanf("%i",&pause);
                printf("%i msec\n", pause);
                if(pause<0 || pause>7000)
                        {
                         printf("\nValue out of range. Please type again.\n");
                         goto marker8;
                        }

                if (pause!=0)
                        {
                                marker9 :printf("\nPlease enter a value for the repetition of the sound source(value must be betwee 1-20):\n");
                                scanf("%i",&round);
```

```
                printf("\nYou have chosen the following number for repetition : %i\n",round);
                if(round<1 || round>20)
                 {
                        printf("\nValue out of range. Please type again.\n");
                        goto marker9;
                 }
        }


for (i=0; i<horizontal; i++)
                        {
                                ywerte[i]= transformation(SOURCEDISTANCE, ((60.0-120.0/(horizontal*2.0))-i*120.0/horizontal*1.0));
                                actualdegy= (atan((ywerte[i]/SOURCEDISTANCE)))*180/PI;
                                printf("\ny=%f inch, y= %f degree",ywerte[i],actualdegy);

                        };

for (i=0; i<vertical; i++)
                        {
                                zwerte[i]= transformation(SOURCEDISTANCE, ((60.0-120.0/(vertical*2.0))-i*120.0/vertical*1.0));
                                actualdegz= (atan((zwerte[i]/SOURCEDISTANCE)))*180/PI;
                                printf("\nz=%f inch, z= %f degree",zwerte[i],actualdegz);

                        };


printf("\n\nDo you want a randomized selection of the window?\n");
jump3: printf("type 'y' for 'yes' and 'n' for 'no':  ");
cin  >> question4;

if (question4!='y'&&question4!='n')
        {
                printf("\nWrong Button! Please type again\n");
                goto jump3;
        }

printf("\nPress any key to start.\n");
key=getch();

if (question4=='y')
        {
                 newround='y';
                 roundcounter=0;
                 r=0;
```

```
                step:
                k=0;
                r=r++;
                printf("\n\n");
                for (i=0; i<vertical; i++)
                        {
                                for (j=0; j<horizontal; j++)
                                        {
                                         vektor1[k].v=letter1[i];
                                         vektor1[k].h=letter2[j];
                                         printf("%c%c,",vektor1[k].v,vektor1[k].h);
                                         k=k++;

                                         };
                                };

                //randomize();
                //repeat=random(5);
                repeat=3;
                j=0;
                step1:
                k=0;
                res=vertical*horizontal;
chkrnd=0;
                step2:
                if (res==0)
                        {
                         index=res;
                        }
                else
                        {
                         srand((unsigned) time(&t));
                         index=rand() % res;
                        }


                if (index==0 && chkrnd==0)
                        {
                         //printf("goto step2");
                         goto step2;
                        }
                chkrnd=1;
                vektor2[k].v=vektor1[index].v;
                vektor2[k].h=vektor1[index].h;
```

```
k=k++;

for (i=index; i<=vertical*horizontal; i++)
            {
                        vektor1[i].v=vektor1[i+1].v;
                        vektor1[i].h=vektor1[i+1].h;
                        };
res=res--;
if (res<0)
           {
            if(j>repeat)
                        {
                         goto exit1;
                        }
            else
                        {
                         for (i=0;i<vertical*horizontal; i++)
                                    {
                                                vektor1[i].v=vektor2[i].v;
                                                vektor1[i].h=vektor2[i].h;
                                                }
                         j=j++;
                         goto step1;
                        }

            }
goto step2;

exit1:
printf("\n\n");
for (j=0; j<vertical*horizontal; j++)
            {
                        printf("%c%c,",vektor2[j].v,vektor2[j].h);
            };
printf("\n\n");
k=0;
s=0;

marker7:

if (newround=='y')
           {
             window[0]=vektor2[k].v;
             window[1]=vektor2[k].h;
             k=k++;
```

```
                s=s++;
               }
printf("\nThe window number is: %c%c\n",window[0],window[1]);
printf("Sound: %i/%i;  Round: %i;\n",s,vertical*horizontal,r);
n=vectorcoordinates(window,horizontal,vertical);

 for (i=0; i<5; i++)
        {
                if (window[0]==letter1[i])
                        {
                         location[AtrnZ]=zwerte[i];
                         actualdegz= (atan((location[2]/SOURCEDISTANCE)))*180/PI;
                         printf("\nKoordinates of the window: z=%2.2f inch, z= %2.2f degree",location[2],actualdegz);
                        }
        }

 for (i=0; i<7; i++)
        {
                if (window[1]==letter2[i])
                        {
                         location[AtrnY]=ywerte[i];
                         actualdegy= (atan((location[1]/SOURCEDISTANCE)))*180/PI;
                         printf("\nKoordinates of the window: y=%2.2f inch, y= %2.2f degree",location[1],actualdegy);
                        }
        }
if (question5=='y')
        {
         if (vertical==2)
                {
                        if (window[0]=='a')
                                {
                                 question3='c';
                                 btron_amplfy_source(SourceID, GAIN_dB_ON+amplification+6);
                                }
                        if (window[0]=='b')
                                {
                                 question3='b';
                                 btron_amplfy_source(SourceID, GAIN_dB_ON+amplification+10);
                                }
                 }

                if (vertical==3)
                        {
if (window[0]=='a')
                                                {
```

```
                                        question3='c';
                                        btron_amplfy_source(SourceID, GAIN_dB_ON+amplification+6);
                                        }
                                if (window[0]=='b')
                                        {
                                        question3='a';
                                        btron_amplfy_source(SourceID, GAIN_dB_ON+amplification);
                                        }
                                if (window[0]=='c')
                                        {
                                        question3='b';
                                        btron_amplfy_source(SourceID, GAIN_dB_ON+amplification+10);
                                        }
                        }

                }


        if (pause==0)
                {
                switch(question3)
                        {
                        case 'a': case 'A': btron_ctrl_wave(SourceID, wave1, WaveCTRL_LOOP, NULL); break;
                        case 'b': case 'B': btron_ctrl_wave(SourceID, wave2, WaveCTRL_LOOP, NULL); break;
                        case 'c': case 'C': btron_ctrl_wave(SourceID, wave3, WaveCTRL_LOOP, NULL); break;
                        case 'd': case 'D': btron_ctrl_wave(SourceID, wave4, WaveCTRL_LOOP, NULL); break;
                        }
                }
        else
                {
switch(question3)
                        {
                        case 'a': case 'A': btron_ctrl_wave(SourceID, wave1, WaveCTRL_STRT, NULL); break;
                        case 'b': case 'B': btron_ctrl_wave(SourceID, wave2, WaveCTRL_STRT, NULL); break;
                        case 'c': case 'C': btron_ctrl_wave(SourceID, wave3, WaveCTRL_STRT, NULL); break;
                        case 'd': case 'D': btron_ctrl_wave(SourceID, wave4, WaveCTRL_STRT, NULL); break;
                        }
                }


        actualdegz= (atan((location[2]/SOURCEDISTANCE)))*180/PI;
        actualdegy= (atan((location[1]/SOURCEDISTANCE)))*180/PI;
        printf("\nLocation of sound source: Y=%6.3f degree,Z=%6.3f degree\n",actualdegy,actualdegz);
```

```
btron_locate_source(SourceID, location);
do {
            btron_update_audio();
            i=i+1;
           } while(u<62000);

btron_update_audio();

if(pause==0)
           {
            btron_update_audio();
            //first = time(NULL);
            btron_update_audio();
            do {
                              btron_update_audio();
                              //second = time(NULL);
                              btron_update_audio();
                  } while((!kbhit()));
           btron_update_audio();
           if (kbhit())
                        {
                              btron_update_audio();
                              getch();
                              btron_update_audio();
                        }
           btron_update_audio();
           }
else
           {
            delay(300);
           }

switch(question3)
                        {
                        case 'a': case 'A': btron_ctrl_wave(SourceID, wave1, WaveCTRL_STOP, NULL); break;
                        case 'b': case 'B': btron_ctrl_wave(SourceID, wave2, WaveCTRL_STOP, NULL); break;
                        case 'c': case 'C': btron_ctrl_wave(SourceID, wave3, WaveCTRL_STOP, NULL); break;
                        case 'd': case 'D': btron_ctrl_wave(SourceID, wave4, WaveCTRL_STOP, NULL); break;
                        }

 roundcounter=roundcounter++;
 if (pause!=0)
            {
                        if (roundcounter<round)
                                    {
```

```
                                        newround='n';

                                        /*first = time(NULL);
                                        do {
                                                second = time(NULL);
                                               } while(difftime(second,first)<=pause);*/

                                        delay(pause);

                                        goto marker7;
                                        }
                                }



                marker11: printf("\nPlease type the window number and press enter!\n");
                scanf("%2s",&window1);

                printf("\nYou have typed the following window nubmer: %2s\n",window1);

                m=vectorcoordinates(window1,horizontal,vertical);
                printf("\nWindow number: %i, Typed Number: %i\n",n,m);
if (m==99)
                        {
                         goto marker11;
                        }

                matrix[n][m]=matrix[n][m]+1;
                printf("\nMatrix: %i",matrix[n][m]);

                printf("\nPlease press any key to proceed or ESC to quit.\n");
                key=getch();
                if (key!=0)
                        {
                         if (key==27)
                                {
                                        goto exit;
                                }
                         if (k>=horizontal*vertical)
                                {
                                        roundcounter=0;
                                        newround='y';
                                        goto step;
                                }
                         roundcounter=0;
```

```
                    newround='y';
                    goto marker7;
                   }

        }
else
        {


                printf("\n");
                ver=vertical;

newround='y';
                roundcounter=0;
                //r=0;
                s=0;

                marker6: printf("\nPlease type a window number and press enter!\n");
                scanf("%2s",&window);



                marker10:
                if (newround=='y')
                        {
                         s=s++;
                        }

                printf("\nThe window number is: %c%c\n",window[0],window[1]);
                printf("Sound: %i/%i;\n",s,vertical*horizontal);

                for (i=1; i<=5; i++)  // vergleicht, ob der erste Buchstabe a,b,c,d oder e ist.
                        {
                                if (ver==i)
                                        {
                                         for(j=0; j<i; j++)
                                                {
                                                 if(window[0]==letter1[j])
                                                        {
                                                         goto out1;
                                                        }
                                                };
                                        }
                        };
```

```
                printf("\nYou have typed a wrong window number. Please type again!!\n");
if (newround=='y')
                        {
                         s=s--;
                        }
                goto marker6;
                out1:
                for (i=1; i<=7; i++)  // vergleicht, ob der zweite Buchstabe 1,2,3,4,5,6 oder 7 ist.
                        {
                                if (horizontal==i)
                                        {
                                         for(j=0; j<i; j++)
                                                {
                                                 if(window[1]==letter2[j])
                                                        {
                                                         goto out2;
                                                        }
                                                };
                                        }
                        };

                printf("\nYou have typed a wrong window number. Please type again!!\n");
                if (newround=='y')
                        {
                         s=s--;
                        }
                goto marker6;
                out2: printf("\nYou have chosen the following window nubmer: %2s\n",window);
                n=vectorcoordinates(window,horizontal,vertical);


                for (i=0; i<5; i++)
                        {
                                if (window[0]==letter1[i])
                                        {
                                         location[AtrnZ]=zwerte[i];
                                         actualdegz= (atan((location[2]/SOURCEDISTANCE)))*180/PI;
                                         printf("\nKoordinaten des Fensters: z=%2.2f inch, z= %2.2f degree",location[2],actualdegz);
                                        }
                        }

                for (i=0; i<7; i++)
                        {
                                if (window[1]==letter2[i])
                                        {
```

```
                                        location[AtrnY]=ywerte[i];
                                        actualdegy= (atan((location[1]/SOURCEDISTANCE)))*180/PI;
                                        printf("\nKoordinates of the window: y=%2.2f inch, y= %2.2f degree",location[1],actualdegy);
                                       }
                    }

                if (pause==0)
                        {
                        switch(question3)
                                {
                                case 'a': case 'A': btron_ctrl_wave(SourceID, wave1, WaveCTRL_LOOP, NULL); break;
                                case 'b': case 'B': btron_ctrl_wave(SourceID, wave2, WaveCTRL_LOOP, NULL); break;
                                case 'c': case 'C': btron_ctrl_wave(SourceID, wave3, WaveCTRL_LOOP, NULL); break;
                                case 'd': case 'D': btron_ctrl_wave(SourceID, wave4, WaveCTRL_LOOP, NULL); break;
                                }
                        }
                else
                        {
switch(question3)
                                {
                                case 'a': case 'A': btron_ctrl_wave(SourceID, wave1, WaveCTRL_STRT, NULL); break;
                                case 'b': case 'B': btron_ctrl_wave(SourceID, wave2, WaveCTRL_STRT, NULL); break;
                                case 'c': case 'C': btron_ctrl_wave(SourceID, wave3, WaveCTRL_STRT, NULL); break;
                                case 'd': case 'D': btron_ctrl_wave(SourceID, wave4, WaveCTRL_STRT, NULL); break;
                                }
                        }


                actualdegz= (atan((location[2]/SOURCEDISTANCE)))*180/PI;
                actualdegy= (atan((location[1]/SOURCEDISTANCE)))*180/PI;
                printf("\nLocation of sound source: Y=%6.3f degree,Z=%6.3f degree\n",actualdegy,actualdegz);

                btron_locate_source(SourceID, location);
                btron_update_audio();
                do {
                        btron_update_audio();
                        i=i+1;
                        } while(u<62000);

                btron_update_audio();

                if(pause==0)
                        {
                        btron_update_audio();
                        //first = time(NULL);
```

```
                btron_update_audio();
                do {
                                btron_update_audio();
                                //second = time(NULL);
                                btron_update_audio();
                        } while((!kbhit()));
                btron_update_audio();
                if (kbhit())
                        {
                                btron_update_audio();
                                getch();
                        }
                btron_update_audio();
              }
else
              {
               delay(400);
              }

switch(question3)
                        {
                        case 'a': case 'A': btron_ctrl_wave(SourceID, wave1, WaveCTRL_STOP, NULL); break;
                        case 'b': case 'B': btron_ctrl_wave(SourceID, wave2, WaveCTRL_STOP, NULL); break;
                        case 'c': case 'C': btron_ctrl_wave(SourceID, wave3, WaveCTRL_STOP, NULL); break;
                        case 'd': case 'D': btron_ctrl_wave(SourceID, wave4, WaveCTRL_STOP, NULL); break;
                        }

 roundcounter=roundcounter++;
 if (pause!=0)
                {
                        if (roundcounter<round)
                                {
                                 newround='n';

                                 /*first = time(NULL);
                                 do {
                                          second = time(NULL);
                                        } while(difftime(second,first)<=pause);*/

                                 delay(pause);

                                 goto marker10;
                                 }
                        }
```

```
marker12: printf("\nPlease type the window number and press enter!\n");
scanf("%2s",&window1);

printf("\nYou have typed the following window nubmer: %2s\n",window1);



m=vectorcoordinates(window1,horizontal,vertical);
printf("\nWindow number: %i, Typed Number: %i\n",n,m);
if (m==99)
        {
         goto marker12;
        }
matrix[n][m]=matrix[n][m]+1;
printf("\nMatrix: %i",matrix[n][m]);

printf("\nPlease press any key to proceed or ESC to quit.\n");
key=getch();
if (key!=0)
        {
         if (key==27)
                {
                        goto exit;
                }
         roundcounter=0;
         newround='y';
         goto marker6;
        }



        };
exit:

printf("\nPlease type a personal ID consisting of 3 letters!\n");
        scanf("%3s",&person);
        printf("\n%s\n",person);
        sprintf(pers,"%s",person);
        printf("\n%s\n",pers);

        if (question5!='y')
                {
                  if (question3=='a' ||question3=='A')
                        {
```

```
                        strcat(pers,"-A.txt");
                        printf("\n%s\n",pers);
                    }
             if (question3=='b' ||question3=='B')
                    {
                        strcat(pers,"-B.txt");
                        printf("\n%s\n",pers);
                    }
             if (question3=='c' ||question3=='C')
                    {
                        strcat(pers,"-C.txt");
                        printf("\n%s\n",pers);
                    }
             if (question3=='d' ||question3=='D')
                    {
                        strcat(pers,"-D.txt");
                        printf("\n%s\n",pers);
                    }
          }
else
          {
           if (vertical==2)
                    {
                     strcat(pers,"-AB.txt");
                     printf("\n%s\n",pers);
                    }
           if (vertical==3)
                    {
                     strcat(pers,"-ABC.txt");
                     printf("\n%s\n",pers);
                    }
          }

pfad="c:\\Gyor2004\\";

strcat(pfad,pers);
printf("\nDatei Name: %s\n",pers);
printf("\nPfad: %s\n",pfad);
referenzdatei = fopen(pfad, "w");

if (referenzdatei==NULL)
          {
          printf("Fehler beim oeffnen der Datei!\n");
          }
```

```
sprintf(satz,"\nVertical Resolution: %i\n",vertical);
fprintf(referenzdatei,"%s\n",satz);

sprintf(satz,"\nHorizontal Resolution: %i\n",horizontal);
fprintf(referenzdatei,"%s\n",satz);



if (question2=='y')
        {
        sprintf(satz,"\nAmplification=%2.1f dB\n",amplification);
        fprintf(referenzdatei,"%s\n",satz);
        }
else
 {
 satz="\nAmplification= default\n";
 fprintf(referenzdatei,"%s\n",satz);
 };
satz="   ";
fprintf(referenzdatei,"%s\n",satz);

satz="\nthe columns represent the real window; the rows represent the given answer;\n";
fprintf(referenzdatei,"%s\n",satz);

satz="   ";
fprintf(referenzdatei,"%s",satz);

for (m=0;m<vertical;m++)
        {
         for (n=0;n<horizontal;n++)
                        {
                         switch (m)
                                        {
                                                case 0: satz="a"; break;
                                                case 1: satz="b"; break;
                                                case 2: satz="c"; break;
                                                case 3: satz="d"; break;
                                                case 4: satz="e"; break;
                                        }
                         switch (n)
                                        {
                                                case 0: satz1="1"; break;
                                                case 1: satz1="2"; break;
                                                case 2: satz1="3"; break;
```

```
                                                case 3: satz1="4"; break;
                                                case 4: satz1="5"; break;
                                                case 5: satz1="6"; break;
                                                case 6: satz1="7"; break;
                                        }

                                fprintf(referenzdatei,"%s%s  ",satz,satz1);
                        }
                }


/*satz="  a1 a2 a3 a4 a5 a6 a7 b1 b2 b3 b4 b5 b6 b7 c1 c2 c3 c4 c5 c6 c7 d1 d2 d3 d4 d5 d6 d7 e1 e2 e3 e4 e5 e6 e7\n";
fprintf(referenzdatei,"%s",satz);*/
i=0;
for (m=0;m<vertical;m++)
                {

                 for (n=0;n<horizontal;n++)
                                        {
                                         satz="\n";
                                         fprintf(referenzdatei,"%s",satz);

                                         switch (m)
                                                {
                                                        case 0: satz="a"; break;
                                                        case 1: satz="b"; break;
                                                        case 2: satz="c"; break;
                                                        case 3: satz="d"; break;
                                                        case 4: satz="e"; break;
                                                }
                                         switch (n)
                                                {
                                                        case 0: satz1="1"; break;
                                                        case 1: satz1="2"; break;
                                                        case 2: satz1="3"; break;
                                                        case 3: satz1="4"; break;
                                                        case 4: satz1="5"; break;
                                                        case 5: satz1="6"; break;
                                                        case 6: satz1="7"; break;
                                                }
                                         /*satz=letter3[m];
                                         satz1=lette4[n];*/

                                         fprintf(referenzdatei,"%s%s",satz,satz1);
                                         for (j=0;j<vertical*horizontal;j++)
```

```
                                        {
                                                fprintf(referenzdatei," %i  ",matrix[i][j]);
                                        }
                                i++;
                                }
                }

 /*     for (i=0;i<vertical*horizontal;i++)
        {

         for (j=0;j<vertical*horizontal;j++)
                        {
                         fprintf(referenzdatei," %i  ",matrix[i][j]);
                        }
        }*/

        fclose(referenzdatei);

         btron_close_wave(wave1);
         btron_close_wave(wave2);
         btron_close_wave(wave3);
         btron_close_wave(wave4);
         btron_close();

        } // end of if(btron_init(Sources,_VERBOSE_) < Ok)

} // end of main()

float  transformation(float quellenabstand, float degree)
{
        float xy;
        xy = quellenabstand*(tan(degree*(PI/180)));
        return xy;
}


int vectorcoordinates(char window1[2],int horizontal, int vertical)
{
int m;
switch (horizontal)
                {
                        case 3: switch (vertical)
                                                                        {
                                                                         case 1: switch (window1[0])
                                                                                                                {
```

```
{
 case '1': m=0; break;
 case '2': m=1; break;
 case '3': m=2; break;
 default: printf("\nWrong Number! Please type again!\n"); m=99;
 };
again!\n"); m=99;
break;
case 2: switch (window1[0])
{
 case '1': m=0; break;
 case '2': m=1; break;
 case '3': m=2; break;
 default: printf("\nWrong Number! Please type again!\n"); m=99;
 };
{
 case '1': m=3; break;
 case '2': m=4; break;
 case '3': m=5; break;
case 'a': switch (window1[1])
break;
default: printf("\nWrong Number! Please type
};
{
case 'a': switch (window1[1])
break;
case 'b': switch (window1[1])
```

```
            default: printf("\nWrong Number! Please type again!\n"); m=99;
            };
again!\n"); m=99;
                                                                    break;
                                                                    case 3: switch (window1[0])
           {
            case '1': m=0; break;
            case '2': m=1; break;
            case '3': m=2; break;
            default: printf("\nWrong Number! Please type again!\n"); m=99;
            };
           {
            case '1': m=3; break;
            case '2': m=4; break;
            case '3': m=5; break;
            default: printf("\nWrong Number! Please type again!\n"); m=99;
            };
           {
                                                                                                                    break;
                                                                                                                    default: printf("\nWrong Number! Please type
                                                                                                                    };
                                                                                                                   {
                                                                                                                    case 'a': switch (window1[1])
                                                                                                                    break;
                                                                                                                    case 'b': switch (window1[1])
                                                                                                                    break;
                                                                                                                    case 'c': switch (window1[1])
```

```
case '1': m=6; break;
case '2': m=7; break;
case '3': m=8; break;
default: printf("\nWrong Number! Please type again!\n"); m=99;
};
```

```
break;
default: printf("\nWrong Number! Please type
again!\n"); m=99;
};
break;
case 4: switch (window1[0])
{
case 'a': switch (window1[1])
{
case '1': m=0; break;
case '2': m=1; break;
case '3': m=2; break;
default: printf("\nWrong Number! Please type again!\n"); m=99;
};
break;
case 'b': switch (window1[1])
{
case '1': m=3; break;
case '2': m=4; break;
case '3': m=5; break;
default: printf("\nWrong Number! Please type again!\n"); m=99;
```

```
        };


       {
        case '1': m=6; break;
        case '2': m=7; break;
        case '3': m=8; break;
        default: printf("\nWrong Number! Please type again!\n"); m=99;
        };


       {
        case '1': m=9; break;
        case '2': m=10; break;
        case '3': m=11; break;
        default: printf("\nWrong Number! Please type again!\n"); m=99;
        };

again!\n"); m=99;
                                                                    break;
                                                                    case 5: switch (window1[0])

       {
        case '1': m=0; break;
                                                                    break;
                                                                    case 'c': switch (window1[1])

                                                                    break;
                                                                    case 'd': switch (window1[1])

                                                                    break;
                                                                    default: printf("\nWrong Number! Please type
                                                                    };

                                                                   {
                                                                    case 'a': switch (window1[1])
```

```
 case '2': m=1; break;
 case '3': m=2; break;
 default: printf("\nWrong Number! Please type again!\n"); m=99;
 };
                                                                                    break;
                                                                                    case 'b': switch (window1[1])
{
 case '1': m=3; break;
 case '2': m=4; break;
 case '3': m=5; break;
 default: printf("\nWrong Number! Please type again!\n"); m=99;
 };
                                                                                    break;
                                                                                    case 'c': switch (window1[1])
{
 case '1': m=6; break;
 case '2': m=7; break;
 case '3': m=8; break;
 default: printf("\nWrong Number! Please type again!\n"); m=99;
 };
                                                                                    break;
                                                                                    case 'd': switch (window1[1])
{
 case '1': m=9; break;
```

```
case '2': m=10; break;

case '3': m=11; break;

default: printf("\nWrong Number! Please type again!\n"); m=99;

};
                                                            break;

                                                            case 'e': switch (window1[1])

{

case '1': m=12; break;

case '2': m=13; break;

case '3': m=14; break;

default: printf("\nWrong Number! Please type again!\n"); m=99;

};
                                                            break;
                                                            default: printf("\nWrong Number! Please type
again!\n"); m=99;
                                                            };
                                        break;
                                        };
                break;

                case 4: switch (vertical)
                                        {
                                        case 1: switch (window1[0])
                                                            {
                                                            case 'a': switch (window1[1])

{

case '1': m=0; break;

case '2': m=1; break;

case '3': m=2; break;
```

```
            case '4': m=3; break;
            default: printf("\nWrong Number! Please type again!\n"); m=99;
            };
again!\n"); m=99;
                                                                                    break;
                                                                                    default: printf("\nWrong Number! Please type
                                                                                    };
                                                                    break;
                                                                    case 2: switch (window1[0])
                                                                                {
                                                                                    case 'a': switch (window1[1])
           {
            case '1': m=0; break;
            case '2': m=1; break;
            case '3': m=2; break;
            case '4': m=3; break;
            default: printf("\nWrong Number! Please type again!\n"); m=99;
            };
                                                                                    break;
                                                                                    case 'b': switch (window1[1])
           {
            case '1': m=4; break;
            case '2': m=5; break;
            case '3': m=6; break;
            case '4': m=7; break;
            default: printf("\nWrong Number! Please type again!\n"); m=99;
```

```
                };

again!\n"); m=99;
                                                                    break;

                                                                    case 3: switch (window1[0])

               {
                case '1': m=0; break;
                case '2': m=1; break;
                case '3': m=2; break;
                case '4': m=3; break;
                default: printf("\nWrong Number! Please type again!\n"); m=99;
                };


               {
                case '1': m=4; break;
                case '2': m=5; break;
                case '3': m=6; break;
                case '4': m=7; break;
                default: printf("\nWrong Number! Please type again!\n"); m=99;
                };
```

```
                                                                    break;
                                                                    default: printf("\nWrong Number! Please type

                                                                    };


                                                                   {
                                                                    case 'a': switch (window1[1])
                                                                    break;

                                                                    case 'b': switch (window1[1])
                                                                    break;

                                                                    case 'c': switch (window1[1])
```

```
{
 case '1': m=8; break;
 case '2': m=9; break;
 case '3': m=10; break;
 case '4': m=11; break;
 default: printf("\nWrong Number! Please type again!\n"); m=99;
 };
again!\n"); m=99;
break;

case 4: switch (window1[0])

{
 case '1': m=0; break;
 case '2': m=1; break;
 case '3': m=2; break;
 case '4': m=3; break;
 default: printf("\nWrong Number! Please type again!\n"); m=99;
 };

{
 case '1': m=4; break;
 break;
 default: printf("\nWrong Number! Please type
 };
{
 case 'a': switch (window1[1])

 break;

 case 'b': switch (window1[1])
```

```
 case '2': m=5; break;
 case '3': m=6; break;
 case '4': m=7; break;
 default: printf("\nWrong Number! Please type again!\n"); m=99;
 };
                                                                    break;
                                                                    case 'c': switch (window1[1])
{
 case '1': m=8; break;
 case '2': m=9; break;
 case '3': m=10; break;
 case '4': m=11; break;
 default: printf("\nWrong Number! Please type again!\n"); m=99;
 };
                                                                    break;
                                                                    case 'd': switch (window1[1])
{
 case '1': m=12; break;
 case '2': m=13; break;
 case '3': m=14; break;
 case '4': m=15; break;
 default: printf("\nWrong Number! Please type again!\n"); m=99;
 };
                                                                    break;
```

```
default: printf("\nWrong Number! Please type again!\n"); m=99;
};
break;
case 5: switch (window1[0])
{
case 'a': switch (window1[1])
{
case '1': m=0; break;
case '2': m=1; break;
case '3': m=2; break;
case '4': m=3; break;
default: printf("\nWrong Number! Please type again!\n"); m=99;
};
break;
case 'b': switch (window1[1])
{
case '1': m=4; break;
case '2': m=5; break;
case '3': m=6; break;
case '4': m=7; break;
default: printf("\nWrong Number! Please type again!\n"); m=99;
};
break;
case 'c': switch (window1[1])
{
```

```
 case '1': m=8; break;
 case '2': m=9; break;
 case '3': m=10; break;
 case '4': m=11; break;
 default: printf("\nWrong Number! Please type again!\n"); m=99;
 };
                                                                                    break;
                                                                                    case 'd': switch (window1[1])
{
 case '1': m=12; break;
 case '2': m=13; break;
 case '3': m=14; break;
 case '4': m=15; break;
 default: printf("\nWrong Number! Please type again!\n"); m=99;
 };
                                                                                    break;
                                                                                    case 'e': switch (window1[1])
{
 case '1': m=16; break;
 case '2': m=17; break;
 case '3': m=18; break;
 case '4': m=19; break;
 default: printf("\nWrong Number! Please type again!\n"); m=99;
```

```
            };

again!\n"); m=99;
                                                                    break;
                                                                    };
                                break;

                                case 5: switch (vertical)
                                                                    {
                                                                     case 1: switch (window1[0])

            {

             case '1': m=0; break;

             case '2': m=1; break;

             case '3': m=2; break;

             case '4': m=3; break;

             case '5': m=4; break;

             default: printf("\nWrong Number! Please type again!\n"); m=99;

             };

again!\n"); m=99;
                                                                     break;

                                                                     case 2: switch (window1[0])

            {

             case '1': m=0; break;
```

```
                                                                    break;
                                                                    default: printf("\nWrong Number! Please type

                                                                    };


                                                                   {
                                                                    case 'a': switch (window1[1])
```

```
                                                                    break;
                                                                    default: printf("\nWrong Number! Please type

                                                                    };


                                                                   {
                                                                    case 'a': switch (window1[1])
```

```
            case '2': m=1; break;
            case '3': m=2; break;
            case '4': m=3; break;
            case '5': m=4; break;
            default: printf("\nWrong Number! Please type again!\n"); m=99;
            };

           {
            case '1': m=5; break;
            case '2': m=6; break;
            case '3': m=7; break;
            case '4': m=8; break;
            case '5': m=9; break;
            default: printf("\nWrong Number! Please type again!\n"); m=99;
            };
again!\n"); m=99;
                                                                    break;
                                                                    case 3: switch (window1[0])
           {
            case '1': m=0; break;
                                                                                                                    break;
                                                                                                                    case 'b': switch (window1[1])
                                                                                                                    break;
                                                                                                                    default: printf("\nWrong Number! Please type
                                                                                                                    };
                                                                                                                   {
                                                                                                                    case 'a': switch (window1[1])
```

```
 case '2': m=1; break;
 case '3': m=2; break;
 case '4': m=3; break;
 case '5': m=4; break;
 default: printf("\nWrong Number! Please type again!\n"); m=99;
 };
                                                                                    break;
                                                                                    case 'b': switch (window1[1])
{
 case '1': m=5; break;
 case '2': m=6; break;
 case '3': m=7; break;
 case '4': m=8; break;
 case '5': m=9; break;
 default: printf("\nWrong Number! Please type again!\n"); m=99;
 };
                                                                                    break;
                                                                                    case 'c': switch (window1[1])
{
 case '1': m=10; break;
 case '2': m=11; break;
 case '3': m=12; break;
 case '4': m=13; break;
```

```
            case '5': m=14; break;
            default: printf("\nWrong Number! Please type again!\n"); m=99;
            };
again!\n"); m=99;
                                                                                    break;
                                                                                    default: printf("\nWrong Number! Please type
                                                                                    };
                                                                    break;
                                                                    case 4: switch (window1[0])
                                                                                                                                    {
                                                                                                                                    case 'a': switch (window1[1])
           {
            case '1': m=0; break;
            case '2': m=1; break;
            case '3': m=2; break;
            case '4': m=3; break;
            case '5': m=4; break;
            default: printf("\nWrong Number! Please type again!\n"); m=99;
            };
                                                                                                                                    break;
                                                                                                                                    case 'b': switch (window1[1])
           {
            case '1': m=5; break;
            case '2': m=6; break;
            case '3': m=7; break;
            case '4': m=8; break;
```

```
 case '5': m=9; break;
 default: printf("\nWrong Number! Please type again!\n"); m=99;
 };
                                                                                    break;
                                                                                    case 'c': switch (window1[1])
{
 case '1': m=10; break;
 case '2': m=11; break;
 case '3': m=12; break;
 case '4': m=13; break;
 case '5': m=14; break;
 default: printf("\nWrong Number! Please type again!\n"); m=99;
 };
                                                                                    break;
                                                                                    case 'd': switch (window1[1])
{
 case '1': m=15; break;
 case '2': m=16; break;
 case '3': m=17; break;
 case '4': m=18; break;
 case '5': m=19; break;
 default: printf("\nWrong Number! Please type again!\n"); m=99;
 };
                                                                                    break;
```

```
again!\n"); m=99;
                                                                    break;
                                                                    case 5: switch (window1[0])
                {
                 case '1': m=0; break;
                 case '2': m=1; break;
                 case '3': m=2; break;
                 case '4': m=3; break;
                 case '5': m=4; break;
                 default: printf("\nWrong Number! Please type again!\n"); m=99;
                 };
                {
                 case '1': m=5; break;
                 case '2': m=6; break;
                 case '3': m=7; break;
                 case '4': m=8; break;
                 case '5': m=9; break;
                 default: printf("\nWrong Number! Please type again!\n"); m=99;
                 };
                                                                    default: printf("\nWrong Number! Please type
                                                                    };
                                                                   {
                                                                    case 'a': switch (window1[1])
                                                                    break;
                                                                    case 'b': switch (window1[1])
                                                                    break;
```

```
                                                                    case 'c': switch (window1[1])
{
 case '1': m=10; break;
 case '2': m=11; break;
 case '3': m=12; break;
 case '4': m=13; break;
 case '5': m=14; break;
 default: printf("\nWrong Number! Please type again!\n"); m=99;
 };
                                                                    break;
                                                                    case 'd': switch (window1[1])
{
 case '1': m=15; break;
 case '2': m=16; break;
 case '3': m=17; break;
 case '4': m=18; break;
 case '5': m=19; break;
 default: printf("\nWrong Number! Please type again!\n"); m=99;
 };
                                                                    break;
                                                                    case 'e': switch (window1[1])
{
 case '1': m=20; break;
 case '2': m=21; break;
```

```
            case '3': m=22; break;
            case '4': m=23; break;
            case '5': m=24; break;
            default: printf("\nWrong Number! Please type again!\n"); m=99;
            };
again!\n"); m=99;
                                                                    break;
                                                                    };
                        break;
                        case 6: switch (vertical)
                                                                    {
                                                                     case 1: switch (window1[0])
          {
            case '1': m=0; break;
            case '2': m=1; break;
            case '3': m=2; break;
            case '4': m=3; break;
            case '5': m=4; break;
            case '6': m=5; break;
            default: printf("\nWrong Number! Please type again!\n"); m=99;
            };
again!\n"); m=99;
                                                                    break;
                                                                    default: printf("\nWrong Number! Please type
                                                                    };
                                                                   {
                                                                    case 'a': switch (window1[1])
                                                                    break;
                                                                    default: printf("\nWrong Number! Please type
                                                                    };
```

```
                                                    break;
                                                    case 2: switch (window1[0])
                                                                                                        {
                                                                                                         case 'a': switch (window1[1])
{
 case '1': m=0; break;
 case '2': m=1; break;
 case '3': m=2; break;
 case '4': m=3; break;
 case '5': m=4; break;
 case '6': m=5; break;
 default: printf("\nWrong Number! Please type again!\n"); m=99;
 };
                                                                                                         break;
                                                                                                         case 'b': switch (window1[1])
{
 case '1': m=6; break;
 case '2': m=7; break;
 case '3': m=8; break;
 case '4': m=9; break;
 case '5': m=10; break;
 case '6': m=11; break;
 default: printf("\nWrong Number! Please type again!\n"); m=99;
 };
                                                                                                         break;
```

```
default: printf("\nWrong Number! Please type
again!\n"); m=99;
};
break;
case 3: switch (window1[0])
{
case 'a': switch (window1[1])
{
case '1': m=0; break;
case '2': m=1; break;
case '3': m=2; break;
case '4': m=3; break;
case '5': m=4; break;
case '6': m=5; break;
default: printf("\nWrong Number! Please type again!\n"); m=99;
};
break;
case 'b': switch (window1[1])
{
case '1': m=6; break;
case '2': m=7; break;
case '3': m=8; break;
case '4': m=9; break;
case '5': m=10; break;
case '6': m=11; break;
default: printf("\nWrong Number! Please type again!\n"); m=99;
```

```
        };

        {
         case '1': m=12; break;
         case '2': m=13; break;
         case '3': m=14; break;
         case '4': m=15; break;
         case '5': m=16; break;
         case '6': m=17; break;
         default: printf("\nWrong Number! Please type again!\n"); m=99;
         };

again!\n"); m=99;
                                                        break;
                                                        case 4: switch (window1[0])

        {
         case '1': m=0; break;
         case '2': m=1; break;
         case '3': m=2; break;
         case '4': m=3; break;
         case '5': m=4; break;
                                                        break;
                                                        case 'c': switch (window1[1])

                                                        break;
                                                        default: printf("\nWrong Number! Please type
                                                        };

                                                       {
                                                        case 'a': switch (window1[1])
```

```
 case '6': m=5; break;
 default: printf("\nWrong Number! Please type again!\n"); m=99;
 };
                                                                                    break;
                                                                                    case 'b': switch (window1[1])
{
 case '1': m=6; break;
 case '2': m=7; break;
 case '3': m=8; break;
 case '4': m=9; break;
 case '5': m=10; break;
 case '6': m=11; break;
 default: printf("\nWrong Number! Please type again!\n"); m=99;
 };
                                                                                    break;
                                                                                    case 'c': switch (window1[1])
{
 case '1': m=12; break;
 case '2': m=13; break;
 case '3': m=14; break;
 case '4': m=15; break;
 case '5': m=16; break;
 case '6': m=17; break;
```

```
            default: printf("\nWrong Number! Please type again!\n"); m=99;
            };
           {
            case '1': m=18; break;
            case '2': m=19; break;
            case '3': m=20; break;
            case '4': m=21; break;
            case '5': m=22; break;
            case '6': m=23; break;
            default: printf("\nWrong Number! Please type again!\n"); m=99;
            };
           {
            case '1': m=0; break;
            case '2': m=1; break;
            case '3': m=2; break;
            case '4': m=3; break;
```

```
                                                                    break;
                                                                    case 'd': switch (window1[1])
                                                                    break;
                                                                    default: printf("\nWrong Number! Please type
                                                                    };
                                                                   {
                                                                    case 'a': switch (window1[1])
```

```
                                                    break;
                                                    case 5: switch (window1[0])
```

```
 case '5': m=4; break;
 case '6': m=5; break;
 default: printf("\nWrong Number! Please type again!\n"); m=99;
 };
                                                                                                    break;
                                                                                                    case 'b': switch (window1[1])
{
 case '1': m=6; break;
 case '2': m=7; break;
 case '3': m=8; break;
 case '4': m=9; break;
 case '5': m=10; break;
 case '6': m=11; break;
 default: printf("\nWrong Number! Please type again!\n"); m=99;
 };
                                                                                                    break;
                                                                                                    case 'c': switch (window1[1])
{
 case '1': m=12; break;
 case '2': m=13; break;
 case '3': m=14; break;
 case '4': m=15; break;
 case '5': m=16; break;
```

```
 case '6': m=17; break;
 default: printf("\nWrong Number! Please type again!\n"); m=99;
 };
                                                                                                                    break;
                                                                                                                    case 'd': switch (window1[1])
{
 case '1': m=18; break;
 case '2': m=19; break;
 case '3': m=20; break;
 case '4': m=21; break;
 case '5': m=22; break;
 case '6': m=23; break;
 default: printf("\nWrong Number! Please type again!\n"); m=99;
 };
                                                                                                                    break;
                                                                                                                    case 'e': switch (window1[1])
{
 case '1': m=24; break;
 case '2': m=25; break;
 case '3': m=26; break;
 case '4': m=27; break;
 case '5': m=28; break;
 case '6': m=29; break;
```

```
            default: printf("\nWrong Number! Please type again!\n"); m=99;
            };
again!\n"); m=99;
                                                                    break;
                                                                    };
                        break;
                        case 7: switch (vertical)
                                                                    {
                                                                     case 1: switch (window1[0])
           {
            case '1': m=0; break;
            case '2': m=1; break;
            case '3': m=2; break;
            case '4': m=3; break;
            case '5': m=4; break;
            case '6': m=5; break;
            case '7': m=6; break;
            default: printf("\nWrong Number! Please type again!\n"); m=99;
            };
again!\n"); m=99;
                                                                    break;
                                                                    case 2: switch (window1[0])
                                                                    {
                                                                     break;
                                                                     default: printf("\nWrong Number! Please type
                                                                     };
                                                                    {
                                                                     case 'a': switch (window1[1])
                                                                     break;
                                                                     default: printf("\nWrong Number! Please type
                                                                     };
```

```
                                                                    case 'a': switch (window1[1])
{
 case '1': m=0; break;
 case '2': m=1; break;
 case '3': m=2; break;
 case '4': m=3; break;
 case '5': m=4; break;
 case '6': m=5; break;
 case '7': m=6; break;
 default: printf("\nWrong Number! Please type again!\n"); m=99;
 };
                                                                    break;
                                                                    case 'b': switch (window1[1])
{
 case '1': m=7; break;
 case '2': m=8; break;
 case '3': m=9; break;
 case '4': m=10; break;
 case '5': m=11; break;
 case '6': m=12; break;
 case '7': m=13; break;
 default: printf("\nWrong Number! Please type again!\n"); m=99;
 };
                                                                    break;
```

```
default: printf("\nWrong Number! Please type
again!\n"); m=99;
};
break;
case 3: switch (window1[0])
{
case 'a': switch (window1[1])
{
case '1': m=0; break;
case '2': m=1; break;
case '3': m=2; break;
case '4': m=3; break;
case '5': m=4; break;
case '6': m=5; break;
case '7': m=6; break;
default: printf("\nWrong Number! Please type again!\n"); m=99;
};
break;
case 'b': switch (window1[1])
{
case '1': m=7; break;
case '2': m=8; break;
case '3': m=9; break;
case '4': m=10; break;
case '5': m=11; break;
case '6': m=12; break;
```

```
            case '7': m=13; break;
            default: printf("\nWrong Number! Please type again!\n"); m=99;
            };

           {
            case '1': m=14; break;
            case '2': m=15; break;
            case '3': m=16; break;
            case '4': m=17; break;
            case '5': m=18; break;
            case '6': m=19; break;
            case '7': m=20; break;
            default: printf("\nWrong Number! Please type again!\n"); m=99;
            };
again!\n"); m=99;
                                                                    break;
                                                                    case 4: switch (window1[0])

           {
            case '1': m=0; break;
            case '2': m=1; break;
                                                                    break;
                                                                    case 'c': switch (window1[1])

                                                                    break;
                                                                    default: printf("\nWrong Number! Please type
                                                                    };

                                                                   {
                                                                    case 'a': switch (window1[1])
```

```
 case '3': m=2; break;
 case '4': m=3; break;
 case '5': m=4; break;
 case '6': m=5; break;
 case '7': m=6; break;
 default: printf("\nWrong Number! Please type again!\n"); m=99;
 };
                                                                                    break;
                                                                                    case 'b': switch (window1[1])
{
 case '1': m=7; break;
 case '2': m=8; break;
 case '3': m=9; break;
 case '4': m=10; break;
 case '5': m=11; break;
 case '6': m=12; break;
 case '7': m=13; break;
 default: printf("\nWrong Number! Please type again!\n"); m=99;
 };
                                                                                    break;
                                                                                    case 'c': switch (window1[1])
{
 case '1': m=14; break;
```

```
                case '2': m=15; break;
                case '3': m=16; break;
                case '4': m=17; break;
                case '5': m=18; break;
                case '6': m=19; break;
                case '7': m=20; break;
                default: printf("\nWrong Number! Please type again!\n"); m=99;
                };
               {
                case '1': m=21; break;
                case '2': m=22; break;
                case '3': m=23; break;
                case '4': m=24; break;
                case '5': m=25; break;
                case '6': m=26; break;
                case '7': m=27; break;
                default: printf("\nWrong Number! Please type again!\n"); m=99;
                };
again!\n"); m=99;
                                                                    break;
                                                                    break;
                                                                    case 'd': switch (window1[1])
                                                                    break;
                                                                    default: printf("\nWrong Number! Please type
                                                                    };
```

```
case 5: switch (window1[0])
{
 case 'a': switch (window1[1])
{
 case '1': m=0; break;
 case '2': m=1; break;
 case '3': m=2; break;
 case '4': m=3; break;
 case '5': m=4; break;
 case '6': m=5; break;
 case '7': m=6; break;
 default: printf("\nWrong Number! Please type again!\n"); m=99;
 };
 break;
 case 'b': switch (window1[1])
{
 case '1': m=7; break;
 case '2': m=8; break;
 case '3': m=9; break;
 case '4': m=10; break;
 case '5': m=11; break;
 case '6': m=12; break;
 case '7': m=13; break;
 default: printf("\nWrong Number! Please type again!\n"); m=99;
```

```
 };
                                                                                                                    break;
                                                                                                                    case 'c': switch (window1[1])
{
 case '1': m=14; break;
 case '2': m=15; break;
 case '3': m=16; break;
 case '4': m=17; break;
 case '5': m=18; break;
 case '6': m=19; break;
 case '7': m=20; break;
 default: printf("\nWrong Number! Please type again!\n"); m=99;
 };
                                                                                                                    break;
                                                                                                                    case 'd': switch (window1[1])
{
 case '1': m=21; break;
 case '2': m=22; break;
 case '3': m=23; break;
 case '4': m=24; break;
 case '5': m=25; break;
 case '6': m=26; break;
 case '7': m=27; break;
```

```
            default: printf("\nWrong Number! Please type again!\n"); m=99;
            };


           {
            case '1': m=28; break;
            case '2': m=29; break;
            case '3': m=30; break;
            case '4': m=31; break;
            case '5': m=32; break;
            case '6': m=33; break;
            case '7': m=34; break;
            default: printf("\nWrong Number! Please type again!\n"); m=99;
            };
again!\n"); m=99;
                                                                    break;
                                                                    };
                        break;
                };
return m;
}
```

```
break;
case 'e': switch (window1[1])
break;
default: printf("\nWrong Number! Please type
};
```