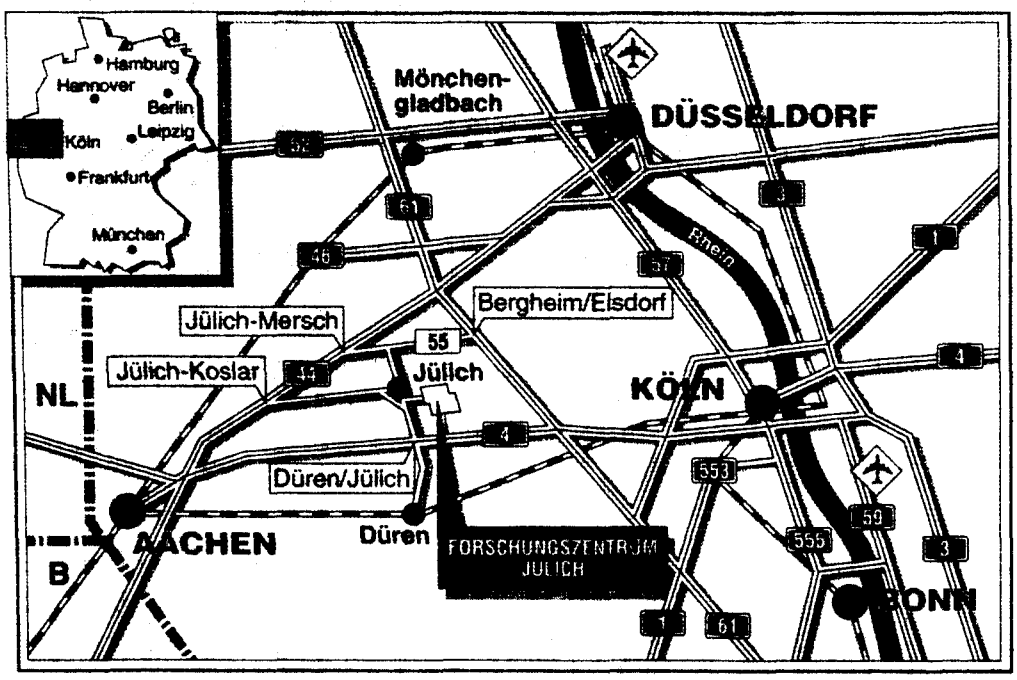
FORSCHUNGSZENTRUM JÜLICH GmbH

Zentralinstitut für Angewandte Mathematik

Parallele Algorithmen und Werkzeuge für die Zuordnung funktionaler und anatomischer Daten auf der Basis von PET- und MRT-Volumendaten

Norbert Esseling



Berichte des Forschungszentrums Jülich ; 2910
ISSN 0944-2952
Zentralinstitut für Angewandte Mathematik Jül-2910

Zu beziehen durch: Forschungszentrum Jülich GmbH · **Zentralbibliothek**
D-52425 Jülich · Bundesrepublik Deutschland
Telefon: 02461/61-6102 · Telefax: 02461/61-6103 · **Telex: 833556-70 kfa d**

Parallele Algorithmen und Werkzeuge für die Zuordnung funktionaler und anatomischer Daten auf der Basis von PET- und MRT-Volumendaten

Norbert Esseling

Kurzfassung

In dieser Arbeit wird ein Verfahren zur räumlichen Zuordnung von funktionaler Information aus der Positronen-Emissions-Tomographie (PET) und anatomischer Information aus der Magnet-Resonanz-Tomographie (MRT) vorgestellt und untersucht. Das hier benutzte Verfahren wird in der Literatur als Kopf-und-Hut-Methode beschrieben. Die Anpassung der Daten erfolgt aufgrund der in den Volumendaten bestimmbaren Kopfoberflächen. Aufgrund dieser Information wird eine Koordinatentransformation berechnet, mit der sich jedem der PET-Datum die entsprechenden höheraufgelösten MRT-Daten zuordnen lassen. Um dem Benutzer eine angenehme Bedienungsoberfläche zu bieten, wird das Visualisierungssystem AVS als Entwicklungs- und Bedienungsumgebung verwendet. Im allgemeinen ist für die Bearbeitung von medizinischen Daten ein hoher Rechenaufwand erforderlich. Um diesem Umstand Rechnung zu tragen, wird im zweiten Teil der Arbeit eine parallele Implementierung des Kopf-und-Hut-Verfahrens auf dem massiv-parallelen System Intel Paragon untersucht. Da die Benutzung einer parallelen Version sich nicht von der sequentiellen Implementierung unterscheiden soll, wird im Rahmen der Arbeit ein Client/Server-Modell entwickelt, das die Vorteile des parallelen Rechnens mit den Vorteilen der Visualisierung und Bedienung unter AVS verbindet.

Abstract

In this report an algorithm to match positron-emission-tomography (PET) scans with nuclear-magnetic-resonance (NMR) scans in three dimensions is presented and evaluated. The algorithm used here is refered as head-and-hat-method in literature. The matching is computed using detectable surfaces in each data set. With this information, a coordinate transform is computed in order to assign PET data and high resolution MRT data. The visualization system AVS is utilized to give the user an easy-to-use interface and is also employed as development environment. As a very high computation demand is expected with this algorithm, the second part of this report describes a parallel implementation of the head-and-hat-method, which was implemented on the massively-parallel computer system Intel Paragon. A client/server model was developed to maintain the same user environment for the sequential and the parallel version of this algorithm. This model combines the advantage of parallel computing with the advantages of visualisation and user interface provided by AVS.

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1

Einleitung

Seit jeher ist der Mensch an der Erforschung seines Inneren interessiert. Neben dem Verständnis der im Menschen ablaufenden Vorgänge spielt die Erkennung von krankhaften Veränderungen eine große Rolle. Vor der Entdeckung der Röntgenstrahlen durch Wilhelm Conrad Röntgen 1895 waren die Untersuchungen der inneren Gewebestrukturen und -funktionen nur an totem Gewebe oder durch sehr riskante Methoden möglich. Seit dieser Zeit ist eine Reihe von technischen Verfahren entwickelt worden, die heute einen Einblick in den lebenden menschlichen Organismus ermöglichen. Einige dieser Techniken sind erst durch den Einsatz der modernen Computertechnik möglich geworden, da die Informationsgewinnung häufig nur mit mathematisch aufwendigen Methoden möglich ist. Die Art der Information hängt hierbei sehr stark von dem jeweils genutzten physikalischen Effekt ab.

Am Institut für Medizin des Forschungszentrums Jülich werden mit Hilfe der Positronen-Emissions-Tomographie (PET) Funktionsweisen wie Blutfluß und Stoffwechsel innerhalb des Kopfes untersucht. Die bei diesem Verfahren gewonnenen Daten geben Aufschluß über die räumliche Verteilung einer injizierten Substanz, die vorher radioaktiv markiert wird. Die Verteilung richtet sich nach der Funktion, die sich mit einer Substanz verfolgen läßt. Wichtiger als die räumliche Verteilung ist aber die Frage nach der Verteilung im Gewebe. Da die Daten der PET nur wenig Information über die jeweiligen Gewebestrukturen enthalten, wird am Institut für Medizin versucht, der funktionalen Information der PET anatomische Information zu überlagern, die aus der Magnet-Resonanz-Tomographie (MRT) gewonnen wird. Auf diese Weise ist eine Zuordnung von Funktion und Struktur möglich. Die Überlagerung der Daten hat sich als aufwendig herausgestellt, da die beiden benutzten Abbildungstechniken Daten mit verschiedenen Auflösungen, Größen und Orientierungen besitzen. Das bisher am Institut für Medizin genutzte Projektionsverfahren [8] zur Überlagerung soll durch das in dieser Arbeit untersuchte und implementierte Kopf-und-Hut-Verfahren ergänzt werden.

Eine Einführung in die Abbildungstechniken PET und MRT, deren Daten innerhalb dieser Arbeit benutzt werden, findet sich in Kapitel 2. In Kapitel 3 wird eine Übersicht und Bewertung bereits in der Literatur existierender Zuordnungsmethoden vor-

genommen. Die Grundlagen für die Implementierung des Kopf-und-Hut-Verfahrens werden in Kapitel 4 behandelt. Da die Handhabbarkeit und Flexibilität für die Benutzung eines Verfahren von entscheidener Bedeutung sind, wurde hier das Visualisierungssystem AVS als Entwicklungsumgebung genutzt. Eine kurze Einführung in dieses System findet sich in Kapitel 5. In Kapitel 6 folgt die sequentielle Implementierung des Zuordnungsverfahrens. Da man bei der hohen Zahl der Daten einen erheblichen Rechenaufwand erwarten kann, wird in Kapitel 8 eine parallele Implementierung des Kopf-und-Hut-Verfahrens beschrieben, die auf dem System Intel Paragon am Zentralinstitut für Angewandte Mathematik des Forschungszentrums Jülich durchgeführt wurde. Um eine einheitliche Bedienung der sequentiellen und parallelen Implementierung zu schaffen, wird in Kapitel 7 ein Client/Server-Modell erarbeitet, das die auf dem parallelen Rechner nicht zur Verfügung stehende Visualisierungsumgebung AVS mit den Vorteilen einer andersartigen Rechnerarchitektur verbindet. Das letzte Kapitel faßt die Arbeit zusammen und gibt einen Ausblick auf weitere Entwicklungen.

Kapitel 2

Medizinische Abbildungsverfahren

In diesem Kapitel wird ein kurzer Einblick in die Arbeitsweise der Abbildungsverfahren, deren Bilddaten in dieser Arbeit verwendet werden, gegeben. Die Positronen-Emissions-Tomographie liefert funktionale Information, während die Magnet-Resonanz-Tomographie aufgrund ihrer physikalischen Eigenschaften eine sehr gute anatomische Gewebedifferenzierung erlaubt. Somit eignen sich die beiden hier vorgestellten Verfahren gut zur Analyse des Gehirns mit dem Ziel, eine Korrelation von Funktion und Anatomie zu erhalten.

2.1 Positronen-Emissions-Tomographie (PET)

Für die Sichtbarmachung von Funktionsweisen des menschlichen Körpers werden radioaktiv markierte Substanzen injiziert, die sich entsprechend den darzustellenden Funktionen im Körper verteilen. Beispiele für zu beobachtende Funktionen sind der Blutfluß oder der Verbrauch bestimmter Substanzen beim Stoffwechsel. Physikalisch beruht dieses Verfahren auf dem Positronenzerfall bei instabilen, neutronenarmen Atomkernen. Unter Abgabe eines Positrons wandelt sich ein Proton des Kerns in ein Neutron um, wie es in Gl. 2.1 beispielhaft für die Umwandlung von Fluor in Sauerstoff beschrieben ist.

$$F_9^{18} \rightarrow O_8^{18} + \beta^+ + \nu \tag{2.1}$$

Die freiwerdende Energie geht dabei auf das Neutrino (ν) und das Positron (β^+) über. Da nicht festgelegt werden kann, welches Teilchen mehr Energie erhält, ergibt sich für das Positron eine kontinuierliche Energieverteilung, deren Maximum bei einem Drittel der maximalen Zerfallsenergie liegt. Das masselose und ungeladene Neutrino fliegt davon, während das Positron in der umgebenden Materie abgebremst

Nuklid	Halbwerts-zeit (min)	max. Energie (MeV)	max. Reichweite (mm H_2O)	max. spez. Aktivität (GBq/mol)
Kohlenstoff-11	20,4	0,97	4,1	$3,4 \times 10^{11}$
Stickstoff-13	9,96	1,19	5,4	$7,0 \times 10^{11}$
Sauerstoff-15	2,05	1,72	8,2	$3,4 \times 10^{12}$
Fluor-18	109,7	0,64	2,4	$6,3 \times 10^{10}$

Tabelle 2.1: Positronenstrahler und deren Eigenschaften [41]

wird. Es vereinigt sich anschließend mit seinem Antiteilchen, dem Elektron, und es entsteht reine Energie, entsprechend den Ruhemassen ($E = mc^2$). Die Energie wird in Form von zwei γ-Quanten zu je 511 keV abgegeben, die sich aufgrund der Impuls- und Energieerhaltung unter einem Winkel von 180° voneinander entfernen. Diese γ-Strahlung kann von außen liegenden Detektoren erfaßt werden. Um sicherzustellen, daß nur Quanten des Positronenzerfallsprozesses registriert werden, stehen sich zwei Detektoren gegenüber. Ein Ereignis wird nur gezählt wird, wenn beide Detektoren innerhalb einer fest vorgegebenen Zeit (typisch einige ns) aktiviert werden. Man bezeichnet diese Form der Zusammenschaltung als Koinzidenz. Tab. 2.1 zeigt exemplarisch einige Positronenstrahler und deren Eigenschaften. Die maximal erreichbare Auflösung wird physikalisch dadurch eingeschränkt, daß der Positronenzerfall nicht an genau demselben Ort liegt wie die Entstehung der γ-Quanten. Das Positron wird nämlich erst nach einer gewissen Flugstrecke stark genug abgebremst und vernichtet. Da zwischen Entstehung und Vernichtung der Positronen einige Stöße mit anderen Teilchen stattfinden können, ist die tatsächliche Reichweite wesentlich geringer als die maximale Reichweite, wie sie aus der Tab. 2.1 hervorgeht. Eine weitere Einschränkung der Auflösung besteht in der Tatsache, daß das Positron im allgemeinen noch nicht völlig zur Ruhe gekommen ist, bevor es vernichtet wird. Dadurch beträgt der Winkel zwischen den γ-Quanten nicht exakt 180°, sondern streut um etwa $0,3°$. Der Ort des Zerfalls liegt aufgrund dieser beiden Effekte nur näherungsweise auf der Verbindungslinie zwischen den beiden Detektoren, so daß schon vom physikalischen Prinzip her keine beliebige Genauigkeit möglich ist. Die physikalische Grenze der Auflösung eines PET wird mit etwa 2–3 mm angegeben [50]; hierin sind jedoch noch nicht die Ungenauigkeiten aufgrund des Abbildungsprozesses enthalten.

Im PET, wie in Abb. 2.1 schematisch dargestellt, sind n Detektoren als Ring angeordnet. Jedem der Detektoren sind jeweils k gegenüberliegende Detektoren in Koinzidenz geschaltet. Dadurch läßt sich für jeden der Detektoren auf dem Ring ein Projektionsprofil ermitteln, woraus dann auf die räumliche Verteilung der Positronenstrahler geschlossen wird. In manchen PET sind mehrere Detektorringe hintereinander eingebaut, um mehrere Schichten des Kopfes gleichzeitig abzutasten. Aus den gemessenen Projektionsdaten lassen sich mit Hilfe geeigneter Rekonstruktionsalgorithmen Schichtbilder berechnen. Die in dieser Arbeit verwendeten Bilder, siehe Abb. 2.2, wurden nach einem iterativen Rekonstruktionsverfahren be-

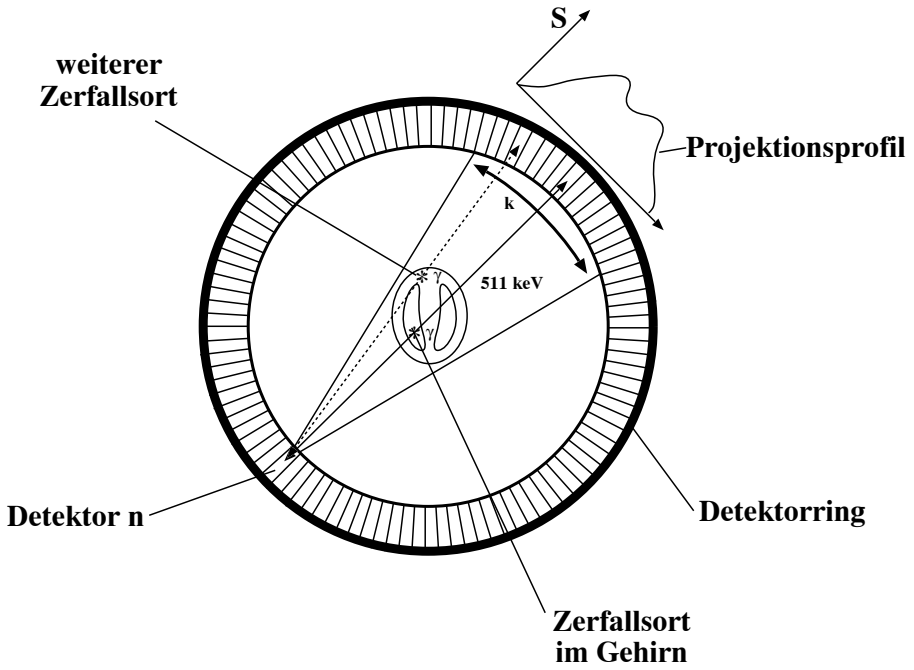


Abbildung 2.1: Schematische Darstellung eines Positronen-Emissions-Tomographen

rechnet [48]. Dabei wird von einer anfangs gleichmäßig initialisierten Bildmatrix jeweils ein Projektionsprofil berechnet und mit dem gemessenen Projektionsprofil verglichen. Anschließend wird die jeweilige Differenz entsprechend der Projektionsrichtung auf die Bildmatrix zurückverteilt und die nächste Projektion berechnet. Sind alle Projektionen einmal durchgerechnet worden, beginnt eine neue Iteration bis eine akzeptable Genauigkeit erreicht ist. Jedem so errechneten Wert wird eine Farbe oder Helligkeit zugeordnet und die Matrix als Schichtbild interpretiert, die die Konzentration des Positronenstrahlers an jedem Punkt der Matrix widerspiegelt.

Bei der PET kann man eine Auflösung von 5–8 mm in transversaler und 6–10 mm in axialer Richtung erwarten. Die hier verwendeten Bilder wie in Abb. 2.2 haben eine Auflösung von etwa 7 mm und werden mit 128×128 Pixeln zu jeweils 2 mm×2 mm dargestellt. Hieraus folgt, daß selbst ein punktförmiges Objekt in mehr als einem Pixel auftaucht.

2.2 Magnet-Resonanz-Tomographie (MRT)

Die MRT beruht auf der von Bloch und Prucell 1946 gefundenen Erscheinung der kernmagnetischen Resonanz. Atomkerne mit ungerader Nukleonenanzahl besitzen eine Asymmetrie und haben ein nicht verschwindendes magnetisches Moment $\vec{\mu}$, das über Gl. 2.2 mit dem Kernspin $\vec{J}$ verbunden ist.

$$\vec{\mu} = \gamma \cdot \vec{J} \tag{2.2}$$

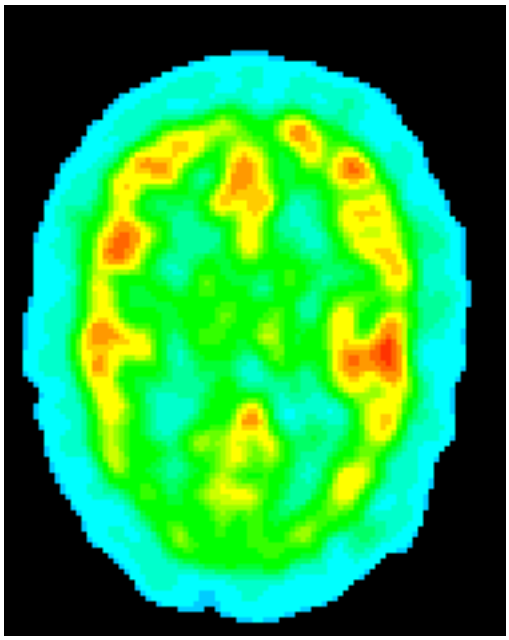


Abbildung 2.2: PET-Schichtbild eines Kopfes

γ stellt das gyromagnetische Verhältnis dar und ist stoffspezifisch. Als Isotop hat neben Kohlenstoff, Natrium und Phosphor der Wasserstoff eine zentrale Bedeutung, da er im Körper relativ häufig und zudem in den verschiedensten Verbindungen vorkommt. Außerdem besitzt der Wasserstoff eine relativ hohe gyromagnetische Konstante γ, die eine hohe Auflösung ermöglicht.

In einem konstanten Magnetfeld mit dem Betrag B_0 richten sich die Momente aus quantentheoretischen Gründen parallel oder antiparallel zum äußeren Feld aus. Dies entspricht zwei diskreten Energiezuständen, bei der die antiparallele Ausrichtung einen energetisch höheren Zustand darstellt. Die Energiedifferenz ΔE beträgt

$$\Delta E = \gamma \cdot \hbar \cdot B_0. \tag{2.3}$$

$\hbar$ ist das Plank'sche Wirkungsquantum. Über einen elektromagnetischen Impuls mit der Frequenz ω kann eine Energie E_{diff} dem Gewebe zugeführt werden. Diese Energie berechnet sich aus

$$E_{diff} = \hbar \cdot \omega. \tag{2.4}$$

Führt man dem Gewebe ein Energiebetrag zu, der der Energiedifferenz zwischen den beiden Momentausrichtungen entspricht, läßt sich leicht auf die Anregungsfrequenz ω_0 (Lamorfrequenz) des Atoms schließen; sie beträgt

$$\omega_0 = \gamma \cdot B_0. \tag{2.5}$$

Für die Magnetisierung eines Volumenelementes gilt die Überlagerung der Einzelmomente jedes Atoms. Der durch Gl. 2.5 beschriebene Effekt ermöglicht die genaue Zuordnung des Betrages eines Magnetfelds an einem Ort zu der zugehörigen

Resonanzfrequenz. Hierdurch lassen sich selektiv Schichten anregen. Wenn die Magnetfeldverteilung nach der Anregung in dem zu untersuchenden Gebiet geschickt gewählt wird, kann man anhand der emittierten Frequenzen auf die Kernverteilung an unterschiedlichen Orten der angeregten Schicht zurückschließen. Dieser Effekt wird für die Bilderzeugung ausgenutzt.

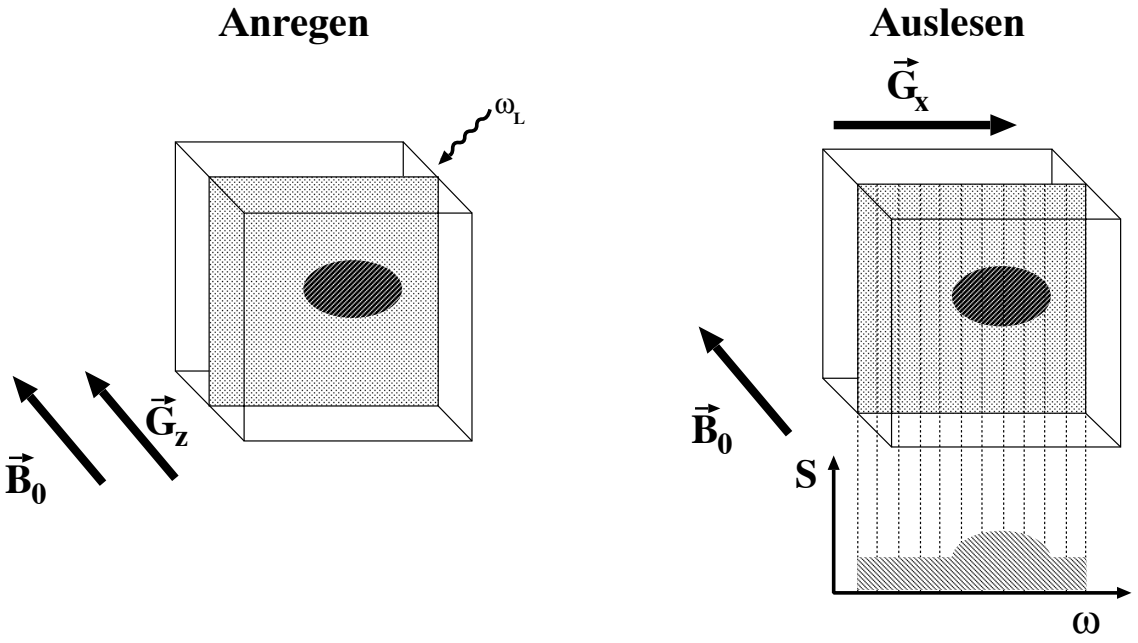


Abbildung 2.3: Frequenzselektives Anregen und Auslesen einer inhomogenen Schicht bei der MRT

Einem homogenen Magnetfeld $\vec{B}_0$ wird in z-Richtung zunächst ein Gradientenfeld $\vec{G}_z$ gleicher Ausrichtung überlagert und so eine Schicht mit der entsprechenden Resonanzfrequenz ω_L angeregt, siehe Abb. 2.3. Da die Kerne ihre Energie wieder in Form von elektromagnetischer Strahlung abgeben, wird das erste Gradientenfeld abgeschaltet und nun ein Gradientenfeld $\vec{G}_x$ quer zur homogenen Magnetfeldrichtung angelegt. Wegen des nun inhomogenen Feldes in der vorher angeregten Schicht wird ein vom Ort abhängiges Frequenzspektrum emittiert. Dieses Frequenzspektrum entspricht der Projektion einer Schicht senkrecht zum Anregungsgradienten. Werden mehrere Projektionen mit Hilfe der Rotation des Gradientenfeldes um die Achse des homogenen Feldes erzeugt, läßt sich mit Hilfe einer geeigneten Rückprojektion in dem betrachteten Fall ein Bild der Protonendichte einer Schicht rekonstruieren.

Aus den Messungen lassen sich neben der Protonendichte ρ auch zwei Relaxationszeitkonstanten T_1 und T_2 ermitteln. T_1 ist die Zeitkonstante für die Longitudinal- oder Spin-Gitter-Relaxation und beschreibt den Vorgang der Rückkehr des Magnetisierungsvektors in die Richtung des statischen Magnetfeldes (z-Richtung) nach einer Anregung. Diese Anregung findet durch einen Puls statt, der den Magnetisierungsvektor gerade um 90° kippt. Er wird deshalb auch 90°-Puls genannt. Die nötige Einstrahldauer wird empirisch ermittelt. Die zeitliche Abhängigkeit des Ma-

gnetisierungsvektors $\vec{M}_z(t)$ in z-Richtung beschreibt Gl. 2.6. $\vec{M}_\infty$ kennzeichnet die Magnetisierung nach Abklingen des Relaxationsprozesses.

$$\vec{M}_z(t) = \vec{M}_\infty \cdot (1 - e^{\frac{-t}{T_1}}) \tag{2.6}$$

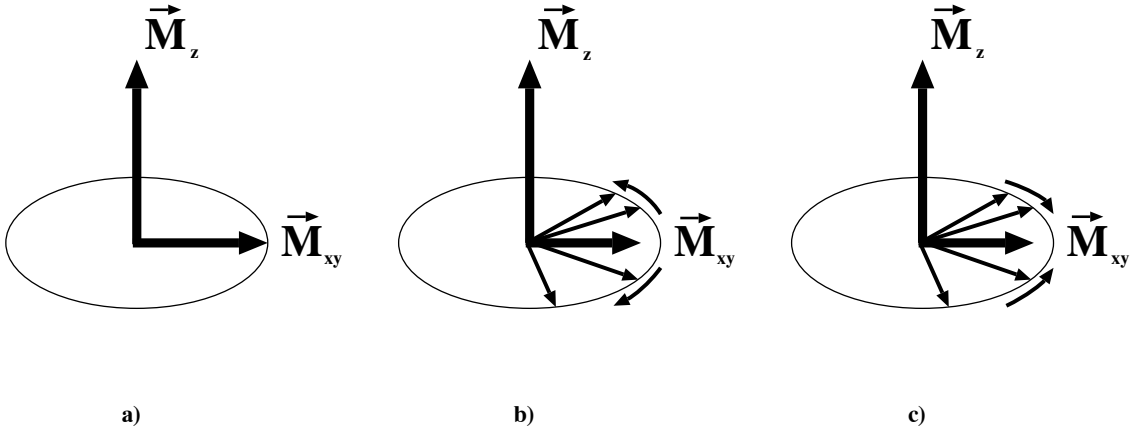


Abbildung 2.4: Magnetisierungsvektor in der xy-Ebene
a) unmittelbar nach dem 90°-Impuls
b) während der Dephasierung
c) nach dem 180°-Impuls (Refokussierung)

Der zweite stattfindende Relaxationsprozeß ist die Transversal- oder Spin-Spin-Relaxation, die durch T_2 charakterisiert wird. Unmittelbar nach der Einstrahlung eines 90°-Impulses rotieren alle Kernspins in Phase, folglich ergibt sich eine maximale, transversale Magnetisierung $\vec{M}_{xy}(0)$ in der xy-Ebene. Anschließend setzt der Dephasierungsprozeß ein, da sich die Phasenlage der Kernspins zu ändern beginnt, wie in Abb. 2.4 gezeigt. Die Magnetisierung $\vec{M}_{xy}(t)$ in der xy-Ebene richtet sich nach der Gl. 2.7. T_2 ist etwa um den Faktor 10 kleiner als T_1.

$$\vec{M}_{xy}(t) = \vec{M}_{xy}(0) \cdot e^{\frac{-t}{T_2}} \tag{2.7}$$

Neben anderen Verfahren [49] verwendet man das sogenannte Spin-Echo-Verfahren zur Messung der einzelnen Parameter. Dazu wird im Zeitabstand T_R (Erholzeit) ein 90°-Impuls eingestreut, nach $T_E/2$ und anschließend nach jeweils T_E (Echozeit) wird ein 180°-Impuls eingestreut. Die 180°-Impulse wirken hier als Ausleseimpulse, da sie die Dephasierung in der xy-Ebene umkehren, siehe Abb. 2.4, und so alle T_E ein Echo erzeugt wird. Man erhält das Maximum des Signals, wenn die reversiblen Anteile der Dephasierung refokussiert sind. Insgesamt folgt die zu erwartende Signalhöhe S der Gl. 2.8 [21], in der sich durch Wahl der Parameter Erholzeit T_R und Echozeit T_E eine gewünschte Gewichtung der Parameter erreichen läßt [49]. c stellt hierbei eine reelle Konstante dar.

$$S = c \cdot \rho \cdot (1 - e^{\frac{-T_R}{T_1}}) \cdot e^{\frac{-T_E}{T_2}} \tag{2.8}$$

Die erreichbaren Auflösungen bewegen sich zwischen 0,5 mm und 2 mm je nach Körperregion und der für diese Region verwendeten Spulen. Abb. 2.5 zeigt einen Ausschnitt eines typisches MRT-Schichtbildes, wobei die Pixelgröße 1 $mm \times$ 1 mm ist. Dargestellt ist hier ein sagittaler Schnitt durch den Kopf eines Patienten. Die Helligkeit ist hierbei ein Maß für die Protonendichte.

Abbildung 2.5: MRT-Schichtbild

Kapitel 3

Räumliche Anpassung von PET- und MRT-Daten

Die räumliche Anpassung von Datensätzen stellt für die Bildverarbeitung ein erhebliches Problem dar. Neben den in dieser Arbeit für den medizinischen Anwendungsfall behandelten Verfahren gibt es weitere Beispiele für viele andere Bereiche wie z.B. Bildkombination bei der Erderkundung mittels Satelliten [19]. Bei den hier vorgestellten Methoden handelt es sich um die Anpassung von Bildern der Abbildungsverfahren MRT und PET. Für die geometrische Anpassung werden die Parameter einer affinen Abbildung des $\mathbb{R}^3$ in den $\mathbb{R}^3$ berechnet. Für diese Art der Abbildung sind neun Parameter zu ermitteln, da die Abbildung die Translation, Rotation und Skalierung in drei Raumrichtungen erlaubt. Die Parameter der Skalierung können jedoch in der Regel aus dem bildgebenden Verfahren mit genügender Genauigkeit abgeschätzt werden. Verzerrungen und Störungen in der Bilderzeugung werden als so gering betrachtet, daß sie hier vernachlässigt werden.

Im folgenden wird eine Reihe von Verfahren vorgestellt, die speziell für den medizinischen Anwendungsfall entwickelt worden sind. Am Ende dieses Kapitels findet ein Vergleich der Verfahren statt.

3.1 Verfahren interner Punkte

Die Idee des Verfahrens 'Interne Punkte' von Evans et al. [10] basiert auf der Tatsache, daß die Anpassung sehr einfach wird, wenn man drei sich jeweils örtlich entsprechende Punkte aus den beiden anzugleichenden Datensätzen kennt. Diese drei Punktepaare genügen zur eindeutigen Bestimmung der Parameter der affinen Transformation, die zur Anpassung führt. Abb. 3.1 zeigt drei solche Punkte in einer ebenen Projektion des beschriebenen Problems. Die sich entsprechenden Punkte sind durch Pfeile verbunden.

Als großes Problem stellt sich die Bestimmung der Punktpaare heraus. Neben den

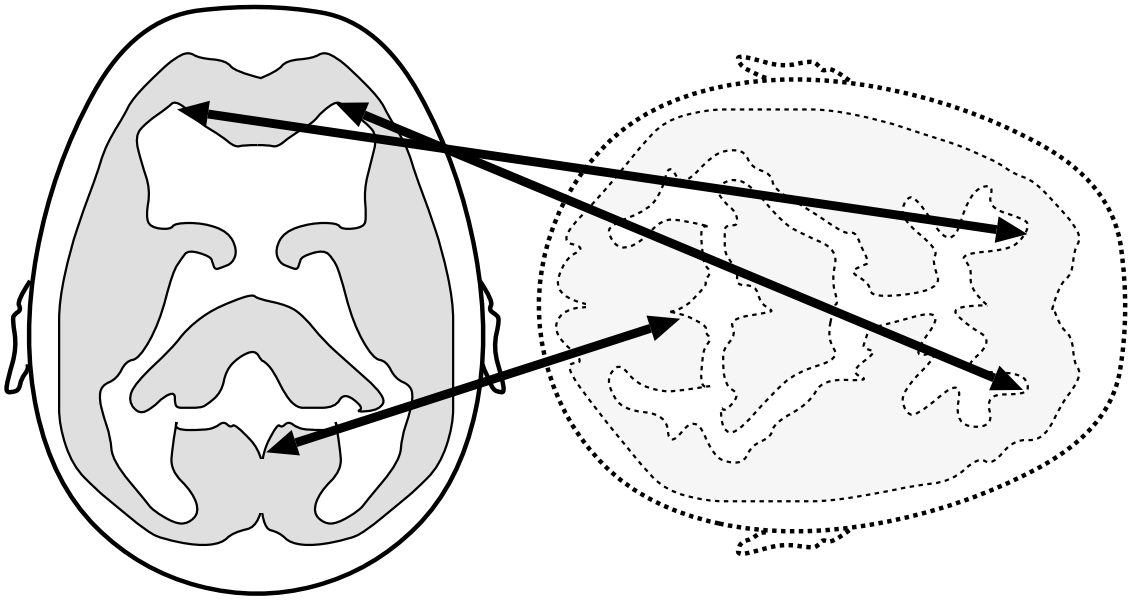


Abbildung 3.1: Angleichung von Datensätzen durch interne sich entsprechende Punkte

verschieden hohen Auflösungen der Daten bereitet die Verschiedenartigkeit der Darstellung Probleme. Evans et al. [10] versuchen diese Problematik durch drei verschiedene Ansätze zu lösen. Als erste Möglichkeit berichten sie über ein Verfahren, bei dem ein Rahmen mit dem Schädel verschraubt wird, auf dem sich Marker für die einzelnen Abbildungsverfahren befinden. Da die Lage der Marker bekannt ist, läßt sich die Transformation in einfacher Weise bestimmen. Auf diese Methode wird jedoch nicht weiter eingegangen, da es sich um ein invasives Verfahren handelt, das dem Patienten nur in Ausnahmefällen zugemutet werden kann. Die zweite Möglichkeit besteht darin, die Punkte interaktiv von einem Experten ermitteln zu lassen. Hierbei findet eine spezielle 3D-Bildverarbeitungs-Software Anwendung, die es dem Betrachter durch Drehungen und Schnitte ermöglicht, mit Hilfe eines Cursors geeignete Punktpaare in den Datensätzen zu bestimmen. Als letztes Verfahren nennen Evans et al. die Möglichkeit, einen bereits vorhandenen Gehirnatlas auf die gemessenen Daten anzupassen. Die Anpassung erfolgt wiederum durch einen Experten, der nun einzelne Schablonen des Hirnatlas auf Hirnregionen der beiden Datensätze anpaßt. Sind die Anpassungen für beide Datensätze bestimmt, so lassen sich daraus in einfacher Weise Punktpaare generieren.

Nach der Bestimmung der 'internen Punkte' benutzen Evans et al. einen Algorithmus zur räumlichen Anpassung der Punkte, der den quadratischen Abstand der Punkte minimiert. Durch die Minimierung können evtl. auftretende Fehler bei der Bestimmung der Punktpaare unterdrückt werden, was durch eine Simulation belegt wird [10]. Schon bei 15 Punktpaaren beträgt der mittlere quadratische Abstand 0,5 mm, wenn die Punkte mit einer Genauigkeit von 5 mm oder weniger angenommen werden. Evans et al. nehmen für die Fehlerverteilung eine Gaußfunktion an.

Der durchschnittliche Positionierungsfehler wird mit 8,8 mm angegeben, wenn die Punkte nur innerhalb der Datensätze ohne weitere Hilfe bestimmt werden. Bei Verwendung eines oben beschriebenen Hirnatlanten in der bereits dargelegten Form sinkt der Positionierungsfehler auf 3,5 mm. Bei diesen Zahlen ist zu berücksichtigen, daß für die Messungen nur ein PET-Scanner mit einer Auflösung von 12 $mm \times$ 12 $mm \times$ 12 mm zur Verfügung stand.

Der Vorteil des Verfahrens liegt in der Stabilität gegen fehlende Daten und im vergleichsweise geringen Rechenaufwand. Ein entscheidender Nachteil ist, daß ein Experte eingesetzt werden muß, um die internen Punkte bestimmen zu können.

3.2 Kopf-und-Hut-Verfahren

Bei dem von Pelizzari et al. [40] entwickelten Verfahren werden die MRT- und PET-Datensätze anhand der in ihnen vorhandenen Oberflächen angeglichen. Für das Verfahren werden zunächst zwei Modelle berechnet, die im folgenden als 'Kopf' und 'Hut' bezeichnet werden. Man erzeugt den Modell-'Kopf' mit Hilfe von automatischen Kontur- und Kantenfindungsverfahren. Für diese Erzeugung findet der Datensatz mit dem größeren Körperausschnitt Verwendung. Sollte der Körperausschnitt in beiden Sätzen gleich sein, benutzt man denjenigen mit der feineren Auflösung. Insgesamt erhält man daraus eine Fläche im dreidimensionalen Raum, die die Außenkontur der abgebildeten Körperregion darstellt. In dem betrachteten Fall der Kombination von PET und MRT verwendet man die Außenkontur des Kopfes im MRT-Bild als Modell-'Kopf'. Das zweite oder 'Hut'-Modell erzeugt man aus dem verbleibenden PET-Datensatz. Hierbei werden einzelne unabhängige Punkte gewählt, die die Außenkontur beschreiben. Die Anzahl der Punkte kann beliebig gewählt werden, ist jedoch in jedem Fall höher als die minimal benötigte Anzahl zur eindeutigen Bestimmung der Parameter für die Transformation. Dies erfordert eine anschließende Optimierung, die dabei zusätzlich eine Minimierung evtl. entstandener Konturfehler ermöglicht. Pelizzari et al. schlagen eine Anzahl von ca. 200 Punkten vor.

Als Maß für die Anpassung verwendet man den mittleren quadratischen Abstand der 'Hut'-Punkte von der 'Kopf'-Fläche. Unter Verwendung eines Minimierungsalgorithmus versucht man, diejenigen Parameter der Transformation zu finden, die angewendet auf die 'Hut'-Koordinaten einen möglichst kleinen quadratischen Abstand ergeben. Der Abstand wird auf der Basis eines Strahles gemessen, der vom transformierten 'Hut'-Punkt zum Zentrum des 'Kopf'-Modells führt.

Abb. 3.2 zeigt auf der linken Seite die Außenkontur eines MRT-Datensatzes, die bei der Anpassung als Fläche im Raum betrachtet wird. Auf der rechten Seite ist ein PET-Datensatz gezeigt, der nur durch die eingezeichneten Außenkonturpunkte definiert ist. Die Punkte des PET-Bildes werden so lange verschoben und gedreht, bis eine Anpassung erreicht ist. Die korrekte Zuordnung der anatomischen MRT-Daten zu einer PET-Schicht errechnet man abschließend durch Interpolation der

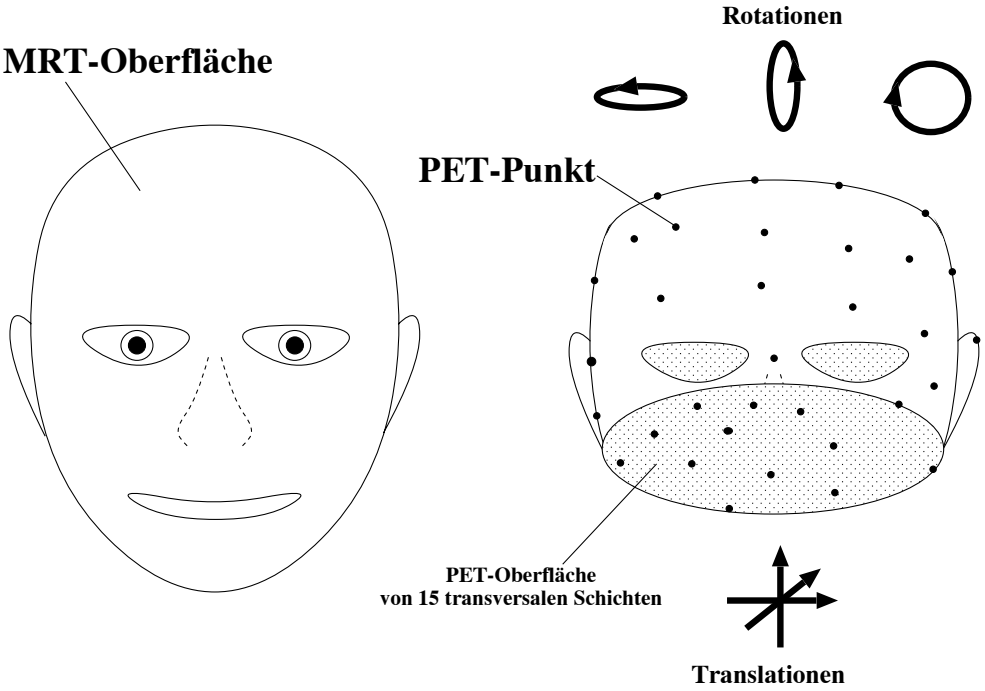


Abbildung 3.2: Angleichung von PET-Punkten an eine MRT-Oberfläche

entsprechenden Schicht im transformierten MRT-Volumendatensatz. Obwohl dieses Verfahren gut dazu geeignet ist, vollkommen automatisch abzulaufen, wird vorgeschlagen, an einigen Stellen manuell einzugreifen. So wird darauf hingewiesen [40], daß der Optimierungsalgorithmus um so besser und vor allen Dingen schneller konvergiert, je genauer die Schätzung für die Startposition der Minimierung ist. Deshalb kann es sinnvoll sein, den Suchalgorithmus zu unterbrechen und einige Parameter von Hand zu ändern, um dann die Suche erneut zu starten. An anderer Stelle wird empfohlen [34], die im wesentlichen automatisch ablaufende Konturfindung manuell zu editieren, um die Ergebnisse zu verbessern. Dies kann besonders im Bereich der Ohren und Nase sinnvoll sein. Wie die Empfehlungen zum manuellen Eingreifen schon zeigen, ist ein Problem der Kopf-und-Hut-Methode eine schnelle Konvergenz. Auf der anderen Seite zeichnet sich dieses Verfahren durch eine relativ hohe Genauigkeit und Robustheit aus. Die Methode dient als Basis für eine Reihe von weiteren Anwendungen [23][36], wobei zum Teil mehr als zwei Abbildungsverfahren aneinander angepaßt werden [20].

3.3 Verwendung von Hauptachsen

Die Methode von N.M. Alpert et al. [5] basiert auf der Annahme, daß ein Körper durch seinen Schwerpunkt und die Orientierung der Hauptachsen im Raum eindeutig in seiner Lage definiert ist. Sind der Schwerpunkt und die Hauptachsen bekannt, so reicht eine einfache Transformation, um die Datensätze aneinander anzupassen.

Zur Schwerpunktbestimmung verwenden sie zwei von Hand erzeugte Oberflächen aus identischen Körperregionen, die in den Volumendaten enthalten sind. Gegenüber der direkten Verwendung der Oberflächen, wie sie bei Gamboa-Aldeco und Chen [15] stattfindet, schlagen Alpert et al. eine gleichmäßige Massedichtefunktion für das gesamte, beschriebene Volumen vor. Nach ihren Angaben wird das Verfahren so wesentlich weniger anfällig gegen experimentelle Fehler. Für die Berechnung des Schwerpunktes ergibt sich:

$$\vec{\bar{x}} = \frac{1}{n} \sum_{i=0}^{n-1} \vec{x}_i. \tag{3.1}$$

mit
$\vec{\bar{x}}$: Koordinaten des Schwerpunktes
n: Anzahl der Punkte des Körpers
$\vec{x}_i = (x_i, y_i, z_i)$: Ortskoordinaten der Körperpunkte

Danach verschiebt man den Koordinatenursprung der Ausgangsdaten in den Schwerpunkt des betrachteten Körpers und erhält mit

$$\vec{\acute{x}}_i = \vec{x}_i - \vec{\bar{x}} \tag{3.2}$$

die neuen Ortkoordinaten der Körperpunkte. Gleichzeitig werden auch alle anderen Punkte um $\vec{\bar{x}}$ geändert. Nach der Verschiebung lassen sich nun die Momente und Produkte der Inertialmatrix I_{in} gemäß Gl. 3.3 berechnen.

$$I_{in} = \begin{pmatrix} \frac{1}{n} \sum_{i=0}^{n-1} \acute{x}_i{}^2 & \frac{1}{n^2} \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} \acute{x}_i \acute{y}_j & \frac{1}{n^2} \sum_{i=0}^{n-1} \sum_{k=0}^{n-1} \acute{x}_i \acute{z}_k \\ \frac{1}{n^2} \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} \acute{x}_i \acute{y}_j & \frac{1}{n} \sum_{j=0}^{n-1} \acute{y}_j{}^2 & \frac{1}{n^2} \sum_{j=0}^{n-1} \sum_{k=0}^{n-1} \acute{y}_j \acute{z}_k \\ \frac{1}{n^2} \sum_{i=0}^{n-1} \sum_{k=0}^{n-1} \acute{x}_i \acute{z}_k & \frac{1}{n^2} \sum_{j=0}^{n-1} \sum_{k=0}^{n-1} \acute{y}_j \acute{z}_k & \frac{1}{n} \sum_{k=0}^{n-1} \acute{z}_k{}^2 \end{pmatrix} \tag{3.3}$$

mit
$\acute{x}_i, \acute{y}_j, \acute{z}_k$: Koordinaten der verschobenen Punkte
n: Anzahl der Punkte

Anschließend läßt sich für jedes Volumen eine Ähnlichkeitstransformation gemäß

$$I_i = S_i I S_i^T \tag{3.4}$$

mit
I: Inertialmatrix im Hauptachsensystem der Datensätze
I_i: Inertialmatrix des Datensatzes i
S_i, S_i^T: Transformationsmatrizen, bestimmt aus den Eigenwerten von I_i

finden. S_i ist hierbei die Rotationsmatrix, welche aus den Eigenvektoren von I_i gewonnen wird. I ist bei gleichem Volumenausschnitt beider Körper gleich, so daß sich die Inertialmatrix I_2 des zweiten Bildes berechnen läßt durch:

$$I_2 = S_2 {S_1}^T I_1 S_1 {S_2}^T \tag{3.5}$$

mit
I_1, I_2: Inertialmatrix für Datensatz 1 und 2
$S_1, {S_1}^T, S_2, {S_2}^T$: Transformationsmatrizen für Satz 1 und 2

Die Anpassung von Datensatz 1 an Datensatz 2 besteht aus einer Translation in den Schwerpunkt und einer anschließenden Rotation mit Hilfe der Rotationsmatrix $S_1 {S_2}^T$.

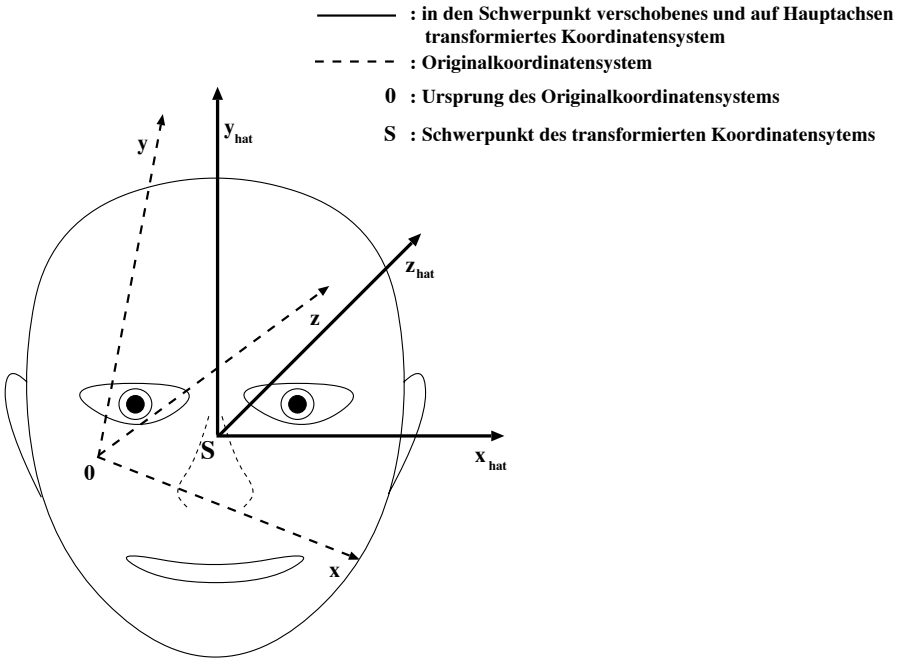


Abbildung 3.3: Transformation auf Hauptachsen

Abb. 3.3 zeigt, wie man sich die oben beschriebene Transformation am Datensatz vorstellen kann. Die Abbildung ist nicht speziell auf MRT- oder PET-Daten bezogen, da die Berechnungen für die Datensätze jeweils gleich sind. Bei der Verwendung von MRT-Daten des menschlichen Gehirns wird für die Bestimmung des Schwerpunktes eine Streuung von kleiner als 0,005 mm und für die Rotation eine Abweichung von $0,16°$ angegeben [5]. Diese Methode ist jedoch anfällig, wenn Daten des Volumens fehlen, wobei zusätzlich noch zu berücksichtigen ist, daß das entsprechende Volumen von Hand selektiert werden muß. Grundsätzliche Probleme treten dann auf, wenn ein hohes Maß an Symmetrie vorhanden ist, weil dann die zu berechnenden Eigenvektoren zum Teil nicht mehr eindeutig sind. Da die Matrizen zur

Transformation in analytischer Weise ermittelt werden können, handelt sich es hier um ein relativ schnelles Verfahren. Aufgrund fehlender PET-Daten konnten Alpert et al. diese Methode jedoch nicht in der Kombination PET und MRT testen.

3.4 Projektionsmethode

Idee des Verfahrens von Huang et al. [24] ist es, die MRT- und PET-Daten aufgrund ihrer Projektion in den drei Raumebenen anzupassen. Der erste grundlegende Schritt ist, wie auch bei anderen Verfahren, die Konturfindung in den verwendeten Datensätzen. Hierzu wird ein relativ einfaches Schwellwertverfahren benutzt. In der Arbeit von Huang et al. [24] werden PET-Transmissionsbilder verwendet, da sie eine gute Differenzierung von Umgebung und Gewebe erlauben. Probleme bereitet jedoch die auf den Bildern befindliche Abbildung des Kopfhalters, die sich automatisch kaum vom Hinterkopf unterscheiden und damit abtrennen läßt. Aus diesem Grund ist in der Regel eine Nachbearbeitung von Hand erforderlich. In der Arbeit von Buchholz [8] wird deshalb vorgeschlagen, iterativ rekonstruierte Bilder zu benutzen, die dort automatisch segmentiert werden. Bei den iterativ rekonstruierten Bildern handelt es sich um Emissionsbilder, die bisher wegen der rechenintensiven Erzeugung kaum Verwendung fanden. Emissionsbilder, die mit dem bislang verwendeten Verfahren der gefilterten Rückprojektion erzeugt wurden, haben ein nicht vernachlässigbares Maß an Artefakten im Außenraum einer Abbildung, die eine Außenkonturbestimmung stark beeinträchtigen. Bei der iterativen Rekonstruktion treten diese Artefakte nahezu nicht mehr auf [48], so daß sich mit diesen Bildern eine Kontur mit dem Schwellwertverfahren bestimmen läßt. Die so berechneten Konturpunkte werden mittels Approximation durch Kreise zu einer Gesamtkontur zusammengefügt [8]. Danach verwendet man das gleiche Verfahren für die Konturfindung in dem MRT-Datensatz. Um nun eine Anpassung vornehmen zu können, werden die erzeugten Konturen der beiden Datensätze in jeweils drei Ebenen projiziert.

Abb. 3.4 zeigt schematisch die Projektionen in der Frontalebene, wobei eine Anpassung durch Drehen und Verschieben der MRT-Kontur erreicht werden soll. Buchholz [8] schlägt zur Anpassung den Standardalgorithmus 'EXTREM' von H.G. Jacob [29] vor, der jeweils die Anpassungsparameter für eine Ebene ermittelt. Nach Ermittlung der Parameter für eine Ebene, wird die sich daraus ergebende neue Projektion berechnet und mit der nächsten Ebene fortgefahren. Man wiederholt dieses Verfahren, bis die Anpassung in allen Ebenen erfolgreich war. Als Gütekriterium findet die Summe des quadratischen Abstandes der Konturen Anwendung. Die Differenz wird auf radialen Strahlen im Abstand von 2° innerhalb der zu vergleichenden Ebenen ermittelt. Den Abschluß des Verfahrens bildet die Neuanordnung der MRT-Daten aufgrund der ermittelten Rotations- und Translationsparameter, sowie die Darstellung der Überlagerung entsprechender Schichten in den PET- und MRT-Volumendaten. Die gefundenen Genauigkeiten liegen bei 2,3 $mm \pm 1{,}1\ mm$ [24][8]. Der Algorithmus hat den Vorteil der automatischen Konturfindung. Probleme tre-

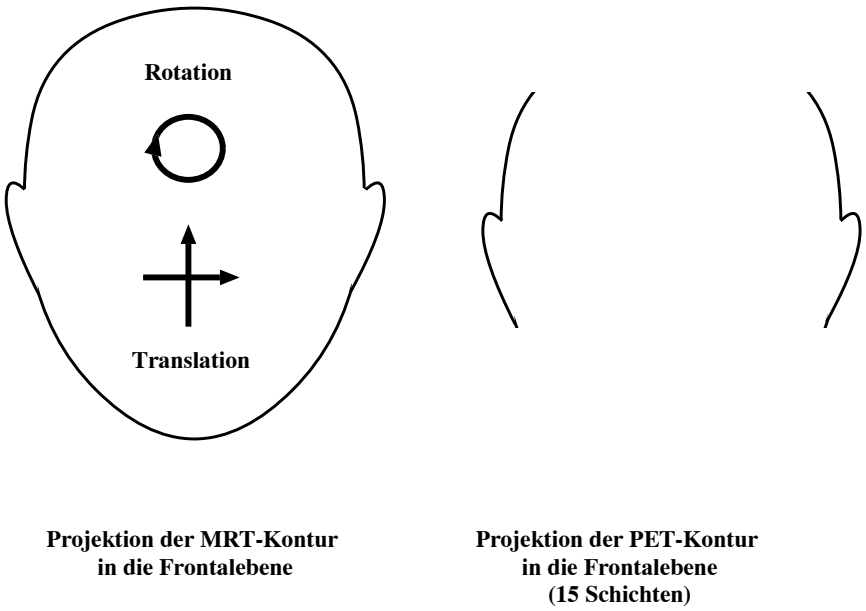


Abbildung 3.4: Projektionen des MRT-Datensatzes und des PET-Datensatzes in die Frontalebene

ten bei diesem Verfahren besonders dann auf, wenn Teile der Außenkontur fehlen. In MRT-Datensätzen fehlt z.B. häufig die Abbildung der Ohren, so daß mit dem oben beschriebenen Verfahren eine Anpassung nur sehr schwer oder gar nicht möglich ist. Zusätzlich treten Probleme auf, wenn Symmetrien innerhalb der Projektionen vorhanden sind. Dies trifft aber ebenso auf andere Verfahren zu.

3.5 Verwendung spezieller Kopfhalter

Kearfott et al. [31] schlagen vor, spezielle Kopfhalter direkt bei der Aufnahme der Daten zu verwenden. Diese Kopfhalter sind für PET wie für MRT verwendbar.

Als Ausgangsmaterial wird eine Polystyrolschale benutzt, die mit einer Latexfolie ausgekleidet wird. Nach Positionierung des Patientkopfes in der Schale bringt man zwischen Folie und Schale Polyurethanschaum, der den Zwischenraum zwischen Kopf und Schale ausfüllt und aushärtet. Der so entstandene Abdruck kann für die Repositionierung des Patientenkopfes verwendet werden, da er sich der Kopfform in idealer Weise angepaßt hat und nur eine Lage des Kopfes im Halter zuläßt. Auf dem so erzeugten Kopfhalter werden nun zusätzlich Marken für eine Laserpositionierung angebracht, um so auch eine genaue Positionierung des Halters in den entsprechenden Tomographen zu ermöglichen.

Obwohl Kopfbewegungen des Patienten nicht völlig ausgeschlossen werden können, verharrt der Kopf unter normalen Umständen in der für ihn bequemsten Position, die durch den Kopfhalter bestimmt ist. Nach den Angaben von Kearfott et al. hat

Methode	Durchschn. Fehler (mm)	Abweichung (mm)	Max. Fehler (mm)
Interne Punkte	1,4	$\pm$0,3	2,8
Kopf-und-Hut	2,2	$\pm$0,5	4,4
Projektion	2,3	$\pm$0,3	-
Hauptachsen	2,1	$\pm$0,7	4,4
spez. Kopfh.[1]	2,0	-	-

Tabelle 3.1: Anpassungsfehler bei verschiedenen Verfahren

der Halter wenig Einfluß auf die jeweiligen Messungen und erzeugt keine Artefakte bei Messung an Phantomen und Patienten. Die Repositionierung gelingt mit einem Lasersystem mit einer Genauigkeit von <2 mm. Da durch die Positionierung die Lage des Kopfes ermittelt werden kann, wird eine einfache Zuordnung der Daten möglich. Über die Anpassung von Datensätzen machen Kearfott et al. hier leider keine Angaben bezüglich der erreichten Genauigkeiten. Ein für die Messung angenehmer Nebeneffekt ist, daß selbst während einer zweistündigen Messung im PET keine Bewegungsartefakte zu verzeichnen sind.

3.6 Vergleichende Übersicht

Die Genauigkeit der Verfahren stellt für diese Arbeit ein wichtiges Kriterium dar. Die dazu benötigte Rechenleistung tritt demgegenüber zunächst in den Hintergrund, da das Verfahren zunächst nicht im klinischen Alltag eingesetzt werden soll, wo die Bearbeitungszeit der Daten eine erhebliche Rolle spielt. Anderson et al. vergleichen in ihrem Artikel [6] die drei hier beschriebenen Verfahren 'Interne Punkte', 'Hauptachsen' und 'Kopf-und-Hut' mittels Simulationen hinsichtlich der erreichbaren Genauigkeit.

Für die Simulation wurden 6 MRT-Datensätze verschiedener Personen zufällig um einen Winkel ($< 30°$) gedreht und verschoben ($<$2,5 mm pro Achse). Aus den so errechneten 6 Datenvolumen wurden anschließend simulierte PET-Scans berechnet. Das PET-Volumen betrug 128$\times$128 Pixel in 35 Schichten. Die einzelnen Volumenelemente hatten dabei eine Größe von 1 $mm\times1$ $mm\times3$ mm und die Auflösung betrug 6 mm. Da die PET-Scans aus den MRT-Daten berechnet wurden, waren die Transformationsmatrizen bekannt, folglich konnte man das Ergebnis der jeweiligen Methode damit vergleichen. Tab. 3.1 zeigt das Ergebnis dieser Untersuchung. Zusätzlich wurden hier die Ergebnisse des Verfahrens 'Projektion' [24][8] aufgenommen.

[1]Hier ist als Fehler nur die Positioniergenauigkeit des verwendeten Lasersystems angegeben.

Das Verfahren der internen Punkten besitzt die höchste Genauigkeit und ist unanfällig gegen fehlende Daten. Die internen Punkte müssen von einem erfahrenen Experten ermittelt werden und sind somit nicht automatisch zu erzeugen. Damit ist es für den in dieser Arbeit betrachteten Anwendungsfall nicht brauchbar. Auch bei der Methode der Hauptachsen ist ein automatischer Ablauf schwer möglich, da exakt der gleiche Volumenausschnitt in beiden Datensätzen gefunden werden muß. Besondere Probleme treten besonders dann auf, wenn einer der Datensätze unvollständig ist. Die Genauigkeit ist im allgemeinen als gut zu bezeichnen, obwohl sie niedriger als bei dem Verfahren der 'Internen Punkte' liegt. Unter Umständen könnte es irgendwann von Vorteil sein, spezielle Kopfhalter zu verwenden, um die Zeit für die Konvergenz der Verfahren zu verkürzen, da so eine gute Schätzung für die Anpassung abgeleitet werden kann.

Das Projektionsverfahren ist in etwa gleich gut wie die 'Kopf-und-Hut'-Methode, jedoch zeigt das Projektionsverfahren Anfälligkeiten gegen fehlende Daten der Außenkontur, wobei es andererseits, wie auch die 'Kopf-und-Hut'-Methode, einen vergleichsweise hohen Rechenaufwandt besitzt. Das 'Kopf-und-Hut'-Verfahren ist von der Konzeption her stabiler, da nicht die Projektion einer Außenkontur auf eine Ebene verwendet wird, sondern eine Anpassung der Kontur im Raum stattfindet. Bei den beiden zuletzt genannten Verfahren ist es vom Prinzip her möglich, sie automatisch zu betreiben, was im Fall der Projektionsmethode schon gezeigt wurde [8]. Insgesamt stehen den Vorteilen dieser beiden Anpassungsalgorithmen und im besonderen der 'Kopf-und-Hut'-Methode ein relativ aufwendiges und rechenintensives Verfahren gegenüber, das zusätzlich mit Konvergenzproblemen belastet sein kann.

Kapitel 4

Grundlagen für die Implementierung

4.1 Bildverarbeitung

Obgleich sich die klassische Bildverarbeitung mit zweidimensionalen Daten beschäftigt, bietet sie eine Reihe von Verfahren für die besprochene Anpassung von medizinischen Volumendaten. Zum Teil können klassische Verfahren direkt angewendet werden, da die Daten in Schichtbildern vorliegen, während an anderer Stelle eine einfache Erweiterung der zweidimensionalen Theorie auf den dreidimensionalen Anwendungsfall möglich ist.

4.1.1 Schwellwertoperation

Eine wichtige Fragestellung der Bildverarbeitung ist, ob ein Bildpixel zu einem Objekt oder zum Bildhintergrund gehört. Bei der Bearbeitung von Bildern wird die zweidimensionale Bildfunktion als Bildmatrix der Größe $n \times m$ dargestellt. Ein Pixel bezeichnet ein Element dieser Bildmatrix und ist in der Regel als ganzzahliger Wert für die Helligkeit des Bildes an der Stelle abgelegt. Farbige Bilder werden in gleicher Weise durch ihre Farbkomponenten dargestellt.

Die Einteilung eines Bildes in Objekte und Hintergrund wird in der Bildverarbeitung als Bildsegmentierung bezeichnet. Neben aufwendigen Verfahren, wie z.B. Textur- und Farbanalysen oder Hough-Transformationen [18][35][43], bildet die Schwellwertoperation eine einfache Möglichkeit, diese Entscheidung zu treffen. Voraussetzung hierfür ist, daß die Pixelwerte sich in zwei Wertebereiche aufteilen lassen, wobei einer dem Objekt und der andere dem Hintergrund zuzuordnen ist. Oft ist das Objekt heller als der Hintergrund, so daß alle Werte oberhalb einer gewissen Grenze als zum Objekt gehörig angesehen werden können. Gl. 4.1 ordnet nach diesem Verfahren jedem Pixel den Wert 1 zu, wenn er zum Objekt gehört und 0, wenn er

zum Hintergrund gehört. Auf diese Weise entsteht ein Binärbild.

$$f_{tres}(x,y) = \begin{cases} 1 & \text{, für } f(x,y) > f_0 \\ 0 & \text{, für } f(x,y) \leq f_0 \end{cases} \tag{4.1}$$

mit
$f(x,y)$: Originalbildmatrix für $x = 0,1,\ldots,x_{max}-1$, $y = 0,1,\ldots,y_{max}-1$
$f_{tres}(x,y)$: Ergebnisbildmatrix für $x = 0,1,\ldots,x_{max}-1$, $y = 0,1,\ldots,y_{max}-1$
x_{max}, y_{max}: Bildausdehnung in x- und y-Richtung
f_0: Schwellwert

Neben der Einteilung in zwei Wertebereiche ist auch eine Einteilung in weitere Bereiche denkbar, um verschiedene Objekte zu trennen. Wählt man eine Zuordnungsfunktion anstelle einer Schwellenfunktion, so ist jede beliebige Zuordnung von altem zu neuem Pixelwert möglich. In diese Kategorie fallen z.B. Kontrastfenster oder Kontrastkorrekturen [35][43].

4.1.2 Konturbeschreibung

Bei vielen Anwendungen ist man daran interessiert, die Information über ein Objekt in möglichst komprimierter Form zu bearbeiten. Hierbei hängt das gewählte Verfahren sehr stark von der Art der zu verwertenden Information ab. In dieser Arbeit ist nur die Oberfläche eines Objektes wichtig, demgegenüber treten andere Objektinformationen in den Hintergrund. Im zweidimensionalen Fall entspricht die Oberfläche eines Körpers dessen Kontur, so daß es nahe liegt, eine Oberfläche eines Volumendatensatz in Schichten zu zerlegen und die darin enthaltenen Konturen zu beschreiben.

Ein Art der Konturbeschreibung sind Kettenkodes [35][43], die den Anfang der Kontur und danach die Lage des nächsten Konturpunktes relativ zum Vorgänger angeben. Abb. 4.1 zeigt ein Beispiel für eine solche Kodierung. Auf der linken Seite ist die Kodierung der Richtungen angegeben, dabei ist jede Richtung in Schritten von 45° durch die Zahlen von 0 bis 7 kodiert. Das Beispielbild zeigt eine Bildmatrix der Größe 9×10. Der Anfangspunkt der gezeigten Kontur ist dunkler als die übrigen Punkte der Kontur markiert. Der unter dem Beispielbild gezeigte Kode beginnt als erstes mit den Anfangspunkten der Kontur, anschließend folgen die Richtungskodes. Neben der sehr einfachen Art der Kodierung bietet dieses Verfahren den Vorteil, typische Merkmale der Kontur zu erkennen. Neben der Ermittlung der Länge der Kontur kann man anhand der Folge der Richtungskodes auf die Konvexität der Kontur schließen [18][35][43].

4.1.3 Koordinatentransformation

Oft ist es notwendig, ein Bild in seiner Orientierung, Lage und Größe zu ändern. Für diesen Fall bedient man sich der affinen Abbildung, die eine Verschiebung, Drehung,

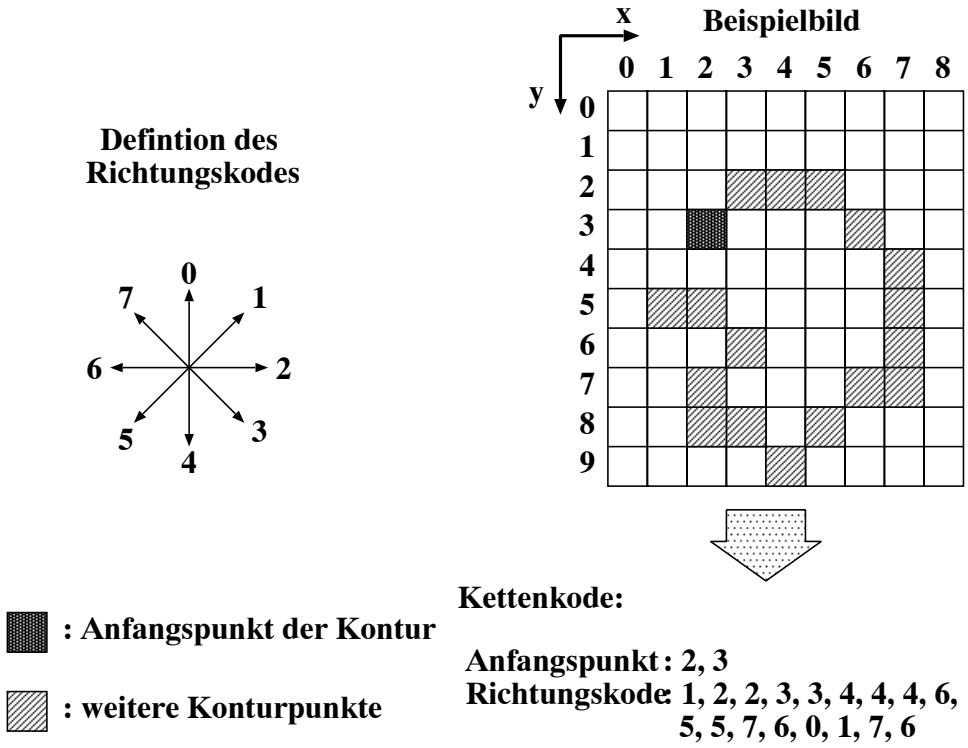


Abbildung 4.1: Konturverfolgung und Kettenkodes

Scherung und Skalierung des Bildes zuläßt. Die Scherung eines Bildes wird nicht weiter in Betracht gezogen, da sie für die Bildtransformation in dieser Arbeit keine Bedeutung hat. Die Rotation um den Winkel ϕ, sowie die Verschiebung $\vec{a}$ läßt sich in der Ebene definieren durch:

$$\vec{x} = \begin{pmatrix} \cos\phi & -\sin\phi \\ \sin\phi & \cos\phi \end{pmatrix} \vec{x} + \vec{a} \tag{4.2}$$

Abb. 4.2 zeigt die Anordnung der Ortsvektoren sowie den gewünschten Drehwinkel ϕ exemplarisch in der xy-Ebene.

Für die in dieser Arbeit beschriebene dreidimensionale Anpassung ist jedoch die Verschiebung und Drehung im Raum nötig. Hierzu lassen sich aus dem zweidimensionalen Fall drei Matrizen ableiten, die jeweils die Drehung um eine der drei Koordinatenachsen beschreiben. Die bei der Drehung um eine Achse nicht veränderte Komponente wird mit einer 1 in der Diagonalen übertragen. Die Matrizen $\mathbf{A_x}$, $\mathbf{A_y}$ und $\mathbf{A_z}$ für die Drehung um die x-, y- und z-Achse sind in Gl. 4.3 dargestellt.

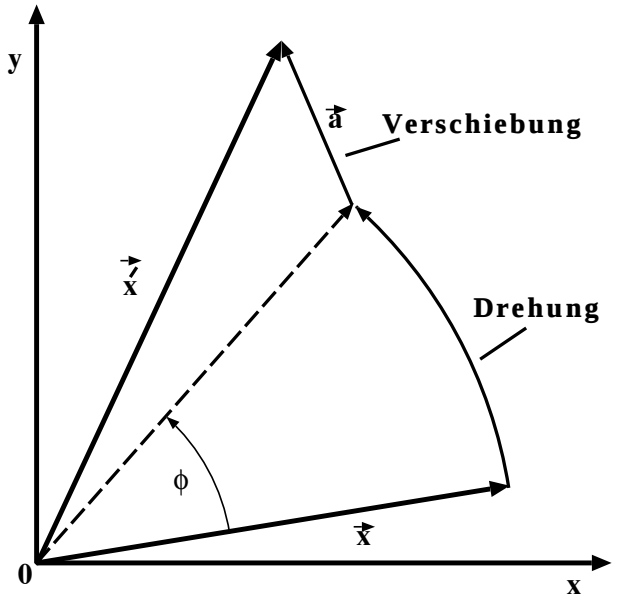


Abbildung 4.2: Drehung und Verschiebung in der xy-Ebene

Zusätzlich ist die Matrix $\mathbf{F}$ zur Skalierung bzgl. der Koordinatenachsen dargestellt:

$$\mathbf{A_x} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\phi_x & -\sin\phi_x \\ 0 & \sin\phi_x & \cos\phi_x \end{pmatrix}, \quad \mathbf{A_y} = \begin{pmatrix} \cos\phi_y & 0 & \sin\phi_y \\ 0 & 1 & 0 \\ -\sin\phi_y & 0 & \cos\phi_y \end{pmatrix},$$

$$\mathbf{A_z} = \begin{pmatrix} \cos\phi_z & -\sin\phi_z & 0 \\ \sin\phi_z & \cos\phi_z & 0 \\ 0 & 0 & 1 \end{pmatrix}, \qquad \mathbf{F} = \begin{pmatrix} f_x & 0 & 0 \\ 0 & f_y & 0 \\ 0 & 0 & f_y \end{pmatrix} \tag{4.3}$$

ϕ_x, ϕ_y und ϕ_z bezeichnen den Winkel der Drehung um die entsprechende Achse, während f_x, f_y und f_z den Vergrößerungsfaktor in Richtung der jeweiligen Achse angeben. Gl. 4.4 zeigt die gesamte Abbildung eines Ortsvektors $\vec{x}$, wobei $\vec{a}$ die Verschiebung des Punktes nach Drehung und Vergrößerung darstellt. Das Ergebnis wird durch $\vec{\acute{x}}$ beschrieben.

$$\vec{\acute{x}} = \mathbf{A_x}\ \mathbf{A_y}\ \mathbf{A_z}\ \mathbf{F}\ \vec{\mathrm{x}} + \vec{\mathrm{a}} \tag{4.4}$$

Diese Gleichung läßt sich auf alle Punkte des zu transformierenden Objekts anwenden und ermöglicht so eine freie 'Bewegung' im Raum. Zusätzlich läßt sich das Objekt in den ursprünglichen drei Raumrichtungen strecken oder stauchen. Hier sei angemerkt, daß die Matrixoperationen nicht kommutativ sind; damit darf die Reihenfolge nicht geändert werden. Zur Verkürzung der Rechenzeit ist es jedoch möglich, die Matrizen $\mathbf{A_x}$, $\mathbf{A_y}$, $\mathbf{A_z}$ und $\mathbf{F}$ zu einer Matrix zusammenzufassen.

4.2 Mathematische Optimierung

Da es für die gewünschte räumliche Anpassung keine eindeutige Lösung gibt, muß eine möglichst gute Anpassung gefunden werden. Die Theorie der Optimierung bietet hier eine Reihe von Verfahren, dieses Problem zu lösen[39][12][16]. Dabei ist es zunächst wichtig, das Optimierungsproblem geeignet zu formulieren.

4.2.1 Problembeschreibung

Im hier betrachteten Fall soll eine Minimierung des Abstandes der die PET-Kontur definierenden Punkte und der MRT-Kontur durchgeführt werden. Die zu optimierenden Größen sind die Parameter der durchzuführenden Koordinatentransformation aus Abschnitt 4.1.3. Die Parameter werden als voneinander unabhängig betrachtet. Die zu minimierende Funktion ist nichtlinear. Damit handelt es sich hier um die Minimierung einer nichtlinearen Funktion mehrerer Variablen ohne Nebenbedingungen.

Mathematisch läßt sich dies durch

$$\min_{\vec{p}\in\mathbb{R}^n} f(\vec{p}) \tag{4.5}$$

beschreiben. Die n zu optimierenden Parameter werden als Komponenten in $\vec{p}$ zusammengefaßt. f ist die zu minimierende Funktion. Für ein analytisch bestimmbares f lassen sich die Extrema und somit auch die Minima durch Suchen der Nullstellen der ersten Ableitung in einfacher Weise finden. Da f in dem vorliegenden Fall eine Funktion der Eingangsdaten und der verwendeten Algorithmen ist, läßt sich keine geschlossene analytische Darstellung finden. Somit ist es nicht möglich, eine Ableitungsfunktion anzugeben, deren Nullstellen sich bestimmen lassen. Die Minimierung wird daher mit rechnergestützten Minimierungsverfahren durchgeführt [39].

4.2.2 Rechnergestützte Minimierung

Rechnergestützte Verfahren sind im allgemeinen iterative Verfahren, für die man zunächst einen beliebigen Startpunkt wählt. Eine gute Schätzung des Startpunktes beschleunigt die Konvergenz des Verfahrens. Anschließend bestimmt man eine Suchrichtung, in der das Minimum gesucht werden soll. Nach der Optimierung der zu minimierenden Funktion entlang der Suchrichtung wird überprüft, ob ein vorzugebendes Abbruchkriterium bereits erreicht ist. Sollte es erreicht sein, kann der Algorithmus abgebrochen werden, im anderen Fall beginnt eine neue Iteration mit der Bestimmung einer neuen Suchrichtung. Die im folgenden vorgestellten Verfahren unterscheiden sich nur in der Wahl der neuen Suchrichtung, die auf jeden Fall eine Abstiegsrichtung darstellen muß. Die Abstiegsbedingung lautet:

$$\vec{s}_i\vec{g}_i < 0 \tag{4.6}$$

$\vec{s_i}$ ist die Suchrichtung und $\vec{g_i}$ der Gradient an der Stelle $\vec{p_i}$. i bezeichnet den Iterationsindex. Sollte der Gradient nicht oder nur schwer analytisch zu bestimmen sein, hilft man sich mit dem Differenzenquotienten aus Gl. 4.7.

$$g_{k,i}(\vec{p_i}) \approx \frac{f(\vec{p_i} + \Delta\vec{p}_{k,i}) - f(\vec{p_i} - \Delta\vec{p}_{k,i})}{2\delta_{k,i}} \tag{4.7}$$

k ist der Index für die Komponente und n die Anzahl der Dimensionen. $\delta_{k,i}$ ist die Schrittweite zur Bestimmung des Quotienten. $\Delta\vec{p}_{k,i}$ ist definiert durch:

$$\begin{pmatrix} 0 \\ \vdots \\ 0 \\ \delta_{k,i} \\ 0 \\ \vdots \\ 0 \end{pmatrix} \tag{4.8}$$

Die Berechnung von $\vec{g_i}$ erfordert $2n$ Funktionsberechnungen von f, wobei die Schrittweite $\delta_{k,i}$ einen nicht zu vernachlässigenden Einfluß auf die Bestimmung des Gradienten hat.

Leider läßt sich kein Verfahren angeben, mit dem sich ein globales Minimum finden läßt. Es sei denn, die Problemstellung ermöglicht dies direkt, wie z.B. bei konvexen Problemstellungen [39]. In allen anderen Fällen bleiben nur die Möglichkeiten, den Startpunkt in Nähe des globalen Minimums zu wählen, die Suche von verschiedenen Punkten zu starten oder spezielle Algorithmen zu nutzen, die die Wahrscheinlichkeit für die Auffindung erhöhen. Besonders im letzten Fall läßt sich bereits vorhandenes Vorwissen über die Funktion geschickt mit einbeziehen.

Achsenparallele Suche

Die achsenparallele Suche stellt ein vergleichsweise einfaches Verfahren dar, da als Suchrichtung $\vec{s}$ abwechselnd eine der Koordinatenrichtungen gewählt wird. In diesem Fall wird nur die Komponente des Gradienten ermittelt, dessen Koordinatenachse man gerade bearbeitet. Man schreitet dann in die negative Richtung der Komponente $g_{k,i}$ des Gradienten an der gerade aktuellen Stelle:

$$\vec{s_i} = \begin{pmatrix} \vdots \\ -\frac{g_{k,i}}{|g_{k,i}|} \\ \vdots \end{pmatrix} \tag{4.9}$$

k ist der Index für die Komponente und i für die Nummer der Iteration. Die Abstiegsbedingung aus Gl. 4.6 wird hierbei erfüllt, da

$$\vec{s_i}\vec{g_i} = -\frac{g_{k,i}}{|g_{k,i}|}g_{k,i} = -|g_{k,i}| < 0, \text{für } g_{k,i} \neq 0 \tag{4.10}$$

gilt. Sollte $g_{k,i} = 0$ sein, fährt man mit der nächsten Koordinatenrichtung fort. Der große Vorteil des Verfahrens liegt darin, daß keine vollständige Berechnung des Gradienten nötig ist, da diese meist einen erheblichen Rechenaufwand darstellt. Abb. 4.3 zeigt das Verfahren graphisch anhand einer zweidimensionalen Funktion, wobei zur Verdeutlichung einige Isolinien, die jeweils einen konstanten Funktionswert darstellen, eingezeichnet sind.

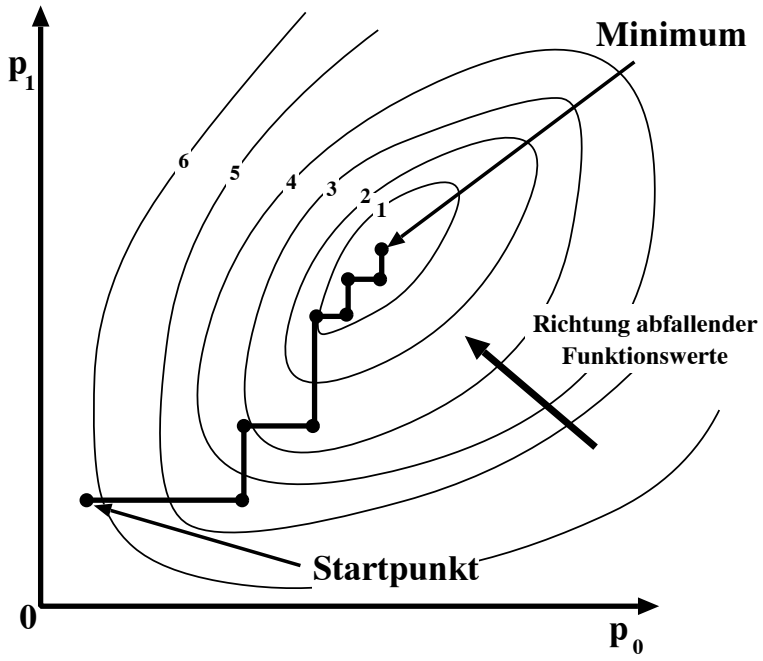


Abbildung 4.3: Achsenparalleles Minimierungsverfahren

Steilster Abstieg

Die Idee dieses Verfahrens ist, durch Suchen in Richtung des steilsten Abstieges das Minimum einer Funktion zu finden. Die negative Gradientenrichtung $\vec{g}_i$ gibt gerade die Suchrichtung $\vec{s}_i$ im Iterationspunkt mit dem Index i vor:

$$\vec{s}_i = -\vec{g}_i. \tag{4.11}$$

Die Abstiegsbedingung Gl. 4.6 ist bei nicht stationären Punkten erfüllt, da gilt:

$$\vec{s}_i \vec{g}_i = -|\vec{g}_i|^2 < 0. \tag{4.12}$$

Dieses Verfahren bietet Vorteile durch eine gleichzeitig stattfindende Suche in allen Komponentenrichtungen, die eine schnellere Konvergenz erwarten läßt.

4.3 Parallelverarbeitung

Mit Hilfe der Parallelverarbeitung kann man den zu erstellenden Anpassungsalgorithmus hinsichtlich der Verarbeitungsgeschwindigkeit deutlich verbessern. In diesem Abschnitt werden die grundlegenden Begriffe und Prinzipien der Parallelverarbeitung erläutert, die in den nachfolgenden Kapiteln benutzt werden.

4.3.1 Parallele Rechnerstrukturen

Die Parallelverarbeitung stellt eine Informationsverarbeitung dar, in der ein Programm nicht mehr sequentiell, d.h. Schritt für Schritt und Datum für Datum, abgearbeitet wird, sondern Instruktionen und/oder Daten zeitgleich bearbeitet werden.

Im weiteren soll eine Rechnerarchitektur aus den folgenden drei grundlegenden Bausteinen bestehen: Kontrolleinheit, Recheneinheit und Speicher. Eine Kontrolleinheit dient der Steuerung vorhandener Recheneinheiten und ist über einen Instruktionspfad (Instruction Streams, I) mit dem Speicher verbunden. Die Recheneinheiten sind über Datenpfade (Data Streams, D) mit dem Speicher verbunden. Nach [13] lassen sich nun vorhandene Rechnerarchitekturen anhand der Vielfachheit der Daten- und Instruktionspfade unterscheiden. Hierbei wird nur unterschieden, ob diese Pfade einfach (single, S) oder mehrfach (multiple, M) vorhanden sind. Unter diesen Bedingungen ergeben sich vier Klassen:

- **SISD:** In diese Klasse fällt die klassische 'von Neumann-Architektur' [22], die nur eine Kontrolleinheit für die Instruktionen und eine Recheneinheit für Daten besitzt.
- **SIMD:** Diese Klasse umfaßt die Vektorrechner, bei denen ein einfacher Instruktionsstrom von Vektorbefehlen vorliegt. Hierbei wird jedes Vektorelement als einzelner Datenstrom betrachtet, so daß ein mehrfacher Datenstrom vorliegt[22].
- **MISD:** Diese Klasse definiert sich durch einen mehrfachen Instruktionsstrom, aber einfachen Datenstrom. Laut [22][25] gibt es keine Beispiele für diese Klasse.
- **MIMD:** Hierbei handelt es sich um die Klasse der Parallelrechner, die eine Reihe von Kontroll- und Verarbeitungseinheiten besitzen. Man hat hier einen mehrfachen Instruktions- und Datenstrom.

Im folgenden werden nur die Rechner der MIMD-Klasse weiter untersucht und verwendet. In [25] wird diese Klasse unterteilt in Systeme mit gemeinsamem Speicher (Shared Memory), wo die einzelnen Prozessoren über ein Verbindungsnetzwerk auf

den gesamten gemeinsamen Speicherraum zugreifen können, und Rechner mit verteiltem Speicher (Distributed Memory). Bei einem System mit verteiltem Speicher hat jede Bearbeitungseinheit (Knoten) einen eigenen lokalen Speicher, in dem sich Daten und Programme befinden. Während Knoten in einem System mit gemeinsamem Speicher Daten über diesen austauschen können, müssen beim System mit verteiltem Speicher Daten von einem Knoten zum anderen auf andere Weise transportiert werden. Zu diesem Zweck besitzen derartige Systeme ein spezielles Verbindungsnetzwerk, das die Knoten untereinander verknüpft. [25] gibt eine Reihe von verschiedenen Topologien für Verbindungsnetzwerke an. Die Anzahl der Knoten kann in einem derartigen System bis zu einigen hundert oder tausend Prozessoren betragen, wobei man dann von einem 'massiv-parallelen System' spricht. Der Austausch von Daten über das Netzwerk erfolgt mit Hilfe von Nachrichten. Man bezeichnet dieses Programmierkonzept als 'Message Passing'. Auf Rechnern, die dieses Konzept unterstützen, sind spezielle Routinen für den Nachrichtenaustausch zwischen den Knoten vorhanden. Zum Teil können so auch komplexere Kommunikationsstrukturen in einfacher Weise realisiert werden. Im weiteren wird nur das Programmierkonzept des verteilten Speichers mit Nachrichtenaustausch betrachtet. Als Programmierstil wurde der sog. 'SPMD'-Stil (Single Program Multiple Data) [9][4] benutzt, der in Verbindung mit dem verwendeten Rechner angeboten wird. Bei dieser Art der Programmierung wird auf jedem Knoten das gleiche Programm gestartet, jedoch arbeitet jeder Knoten auf anderen Daten des Gesamtproblems.

4.3.2 Begriffe der Parallelverarbeitung

Bei einem parallelen Programm werden Teilaufgaben des Programms zeitgleich auf mehreren Knoten bearbeitet. Für diese Teilaufgaben lassen sich die in einem Problem vorhandenen **Datenparallelitäten** und **Funktionsparallelitäten** nutzen. Datenparallelität liegt dann vor, wenn mehrere Daten unabhängig voneinander bearbeitet werden können. Von Funktionsparallelität spricht man, wenn zeitgleich verschiedene Funktionen ausgeführt werden können. Um einen korrekten Ablauf eines Programms zu gewährleisten, muß unter Umständen eine **Synchronisation** der Teilaufgaben stattfinden.

Im folgenden wird eine Reihe von Begriffen definiert, die einen Vergleich eines parallelen Programms gegenüber einer sequentiellen Programmierung ermöglichen. Einen direkten Vergleich bietet der Begriff **Speedup**. Hierbei wird das Verhältnis aus der Ausführungszeit des sequentiellen Programms bezogen auf die Ausführungszeit des entsprechenden parallelen Programms betrachtet. Die obere Grenze für diesen Speedup liegt bei der Zahl der jeweils benutzten Knoten. In der Regel ist der Speedup jedoch nicht maximal, da, verglichen mit dem sequentiellen Programm, ein zusätzliches Maß an Verwaltung notwendig ist, um die Arbeit der Knoten zu koordinieren. Häufig ist für diese Koordinierung **Kommunikation** notwendig. Kommunikation ist in der Regel besonders zeitaufwendig, verglichen mit der in dieser Zeit möglichen Anzahl an Rechenoperationen. Auch eine ungleiche **Lastverteilung** wirkt sich in diesem Zusammenhang ungünstig aus, da die Knoten zu unter-

schiedlichen Zeiten mit Teilaufgaben fertig werden und dann beim Austausch von Ergebnissen auf Knoten warten müssen, die noch beschäftigt sind.

Das Verhältnis von erreichtem Speedup zu maximal möglichen Speedup (Knotenzahl) wird als **Effizienz** bezeichnet. Eine hohe Effizienz deutet auf eine gute Parallelisierbarkeit eines Problemes hin.

Ein System wird als gut **skalierbar** bezeichnet, wenn die erzielte Leistung mit zunehmender Knotenzahl weiter anwächst.

Kapitel 5

AVS als Entwicklungsumgebung

Das Visualisierungssystem AVS wird in dieser Arbeit als Basis für die Programmentwicklung verwendet. Das modulare Konzept von AVS erweist als sehr hilfreich, da es neben der einfachen Überprüfung eines Programmoduls durch dessen festgelegte Schnittstellen auch eine unabhängige Benutzung der erstellten Module zuläßt. Die zu dem System gehörenden Darstellungsmöglichkeiten und angebotenen Bedienungselemente lassen sich dabei gut für eine Benutzerschnittstelle verwenden.

5.1 Allgemeines zum Application Visualization System AVS

AVS ist die Abkürzung für Application Visualization System und eine von der Firma Advanced Visual System Inc. vertriebene Software. In der vorliegenden Arbeit wurde AVS Version 5 auf der UNIX Workstation Stardent GS 2002 verwendet. Es handelt sich um ein System, das anfangs für die wissenschaftliche Visualisierung gedacht war. Während AVS Version 1 nur aus einem sehr leistungsfähigen Werkzeug zur Darstellung von Geometrien bestand, besaß AVS Version 2 schon einen Netzwerk-Editor, der die Zusammenstellung von Visualisierungsnetzwerken aus einer Reihe von 60 Modulen ermöglichte. Die folgenden Versionen bis hin zur derzeitigen Version 5 bieten neben der Funktionalität der Vorläufer eine Reihe von neuen Funktionen, die besonders dem Entwickler von Visualisierungsprogrammen entgegenkommen. Die Anzahl der Module ist dabei auf 230 angestiegen. AVS bietet neben den bereits zahlreich vorhandenen Modulen die Möglichkeit, eigene Module zu erstellen, die mit im System enthaltenen Komponenten gekoppelt werden können. Auf diese Weise läßt sich jeder gewünschte Algorithmus in die AVS-Umgebung integrieren und mit den vorhandenen Elementen von AVS bedienen.

5.2 Prinzipien von AVS

Die grundlegende Berechnungseinheit von AVS ist das Modul. Ein Modul bearbeitet die eingehenden Daten und erzeugt entsprechende Ausgangsdaten, die wiederum anderen Modulen zugeführt werden können. Abhängig von der Funktion werden die Module in vier Kategorien eingeteilt:

- **Data Input:** Module, die Daten generieren oder Daten von außerhalb einer AVS-Umgebung importieren und in AVS-Datentypen umwandeln.
- **Filter:** Module, die Daten im AVS-Datentyp bearbeiten und wieder als AVS-Datentyp ausgeben.
- **Mapper:** Module, die AVS-Daten in AVS-Geometriedaten umwandeln.
- **Data Output:** Module, die AVS-Daten in der endgültigen Form auf dem Bildschirm ausgeben oder auf anderen Geräten ablegen.

Nach dem Datenflußprinzip können die Ausgänge und Eingänge der Module miteinander verbunden werden, auf diese Weise entsteht ein Netzwerk aus Modulen, die dann eine AVS-Applikation bilden, vgl. Abschnitt 5.4 . Die Ein- und Ausgänge der Module sind farbig kodiert entsprechend den Daten die dort erzeugt oder angenommen werden. Eine Übersicht zur Kodierung findet sich in [3].

Die zu bearbeitenden Daten gelangen an den Eingabemodulen (Data Input) in das Netzwerk und werden anschließend entlang den vorher festgelegten Verbindungen innerhalb des Netzes zu den bearbeitenden Modulen (Filter, Mapper) geführt. Der Datenstrom endet an den Datenausgabemodulen (Data Output), wo er zur Anzeige kommt oder abgelegt wird. Da die einzelnen Module als separate Prozesse innerhalb eines Rechners gestartet werden, enthält der Systemkern des AVS eine Datenflußkontrolle, die die einzelnen Datenflüsse der Module koordiniert. Hierzu ist der Flußkontrolle der Aufbau des Netzwerkes bekannt. Aus dem Aufbau des Netzes ergeben sich die Datenabhängigkeiten der Module und damit die Ablaufreihenfolge und die nötige Datenzuführung.

Ein Beispiel für ein modulares Netzwerk findet sich in Abb. 5.1. Das Modul *read field* liest einen Datensatz im AVS-Format. Diese Daten werden anschließend dem Modul *orthogonal slicer* übergeben, das nach Vorgabe der Schichtnummer eine Schicht aus dem Datensatz herausschneidet. Die nötigen Parameter erhält es dabei von der Benutzeroberfläche, vgl. Abschnitt 5.3 . Das Modul *colorizer* färbt die aus dem Modul *orthogonal slicer* bezogenen Daten mit Hilfe der Farbtafel ein, die er aus dem Modul *generate colormap* erhalten hat. Die so eingefärbte Schicht wird mit dem Modul *display image* auf dem Bildschirm dargestellt.

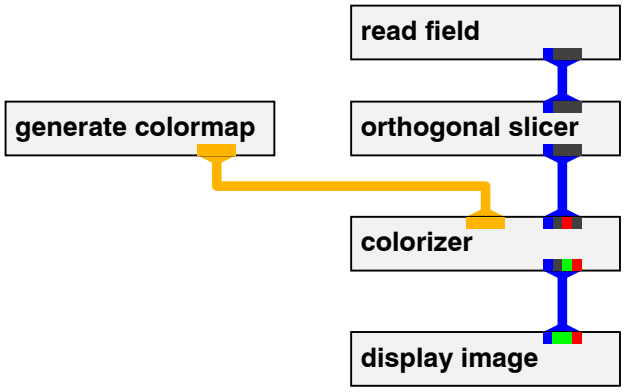


Abbildung 5.1: AVS-Beispielnetzwerk

5.3 Benutzeroberfläche

Als Benutzeroberfläche wird die Oberfläche bezeichnet, wie sie sich dem Benutzer einer Applikation bietet. Sie stellt die Schnittstelle zwischen System und Benutzer dar. Die graphische Darstellung stützt sich hierbei auf den bei UNIX-Systemen häufig verwendeten Standard 'X'. AVS bietet für die Benutzeroberfläche eine Reihe von Bedienungselementen an. Im folgenden werden einige dieser Elemente erklärt, weitere finden sich in Abschnitt 5.5 .

Eine Benutzeroberfläche für das Netzwerk aus Abb. 5.1 ist als Beispiel in Abb. 5.2 dargestellt. Jedes Bedienungselement läßt sich einem Modul aus dem Beispielnetzwerk zuordnen. Mit dem Element *Read Field Browser* läßt sich der Dateiname des gewünschten Datensatzes auswählen, während man mit dem Drehzeiger *slice plane* die gewünschte Schichtnummer vorgeben kann. Der Endanschlag dieses Zeigers wird dabei automatisch der maximalen Anzahl der in dem Datensatz vorhandenen Schichten angepaßt. Ganz rechts befindet sich das Bedienfeld *colormap* zur Erstellung von Farbtafeln für die Einfärbung der Schichten. Die Schalter dieses Feldes dienen dem Editieren der Farbtafel. Die Zuordnung der Farben geschieht mit dem Modul *colorizer*, wobei jedem Datenwert eine Farbe aus der interaktiv erzeugten Farbtafel zugeordnet wird. Das obere Ende der Tafel stellt den Farbwert für den Funktionswert 0 dar, während das untere Ende die Farbe für den Maximalwert darstellt. Der Maximalwert ist in diesem Fall auf 255 eingestellt. Die Funktionseinheit *display image* dient der Ausgabe der Bilder auf dem Bildschirm. Das Beispielbild zeigt eine Schicht aus einem PET-Datensatz.

Strategie der Datenflußkontrolle ist es, nur die Teile eines Netzwerkes zu bearbeiten, die bei einer Daten- oder Parameteränderung gemäß den Datenabhängigkeiten bearbeitet werden müssen. Wenn man in Abb. 5.1 die Farbtabelle ändern würde, würden aus diesem Grunde nur die Module *generate colormap*, *colorizer* und *display image* bearbeitet. Bei den Modulen *read field* und *orthogonal slicer* ändern sich in diesem Fall weder Eingangsdaten noch Parameter, folglich müssen sie nicht bearbeitet werden.

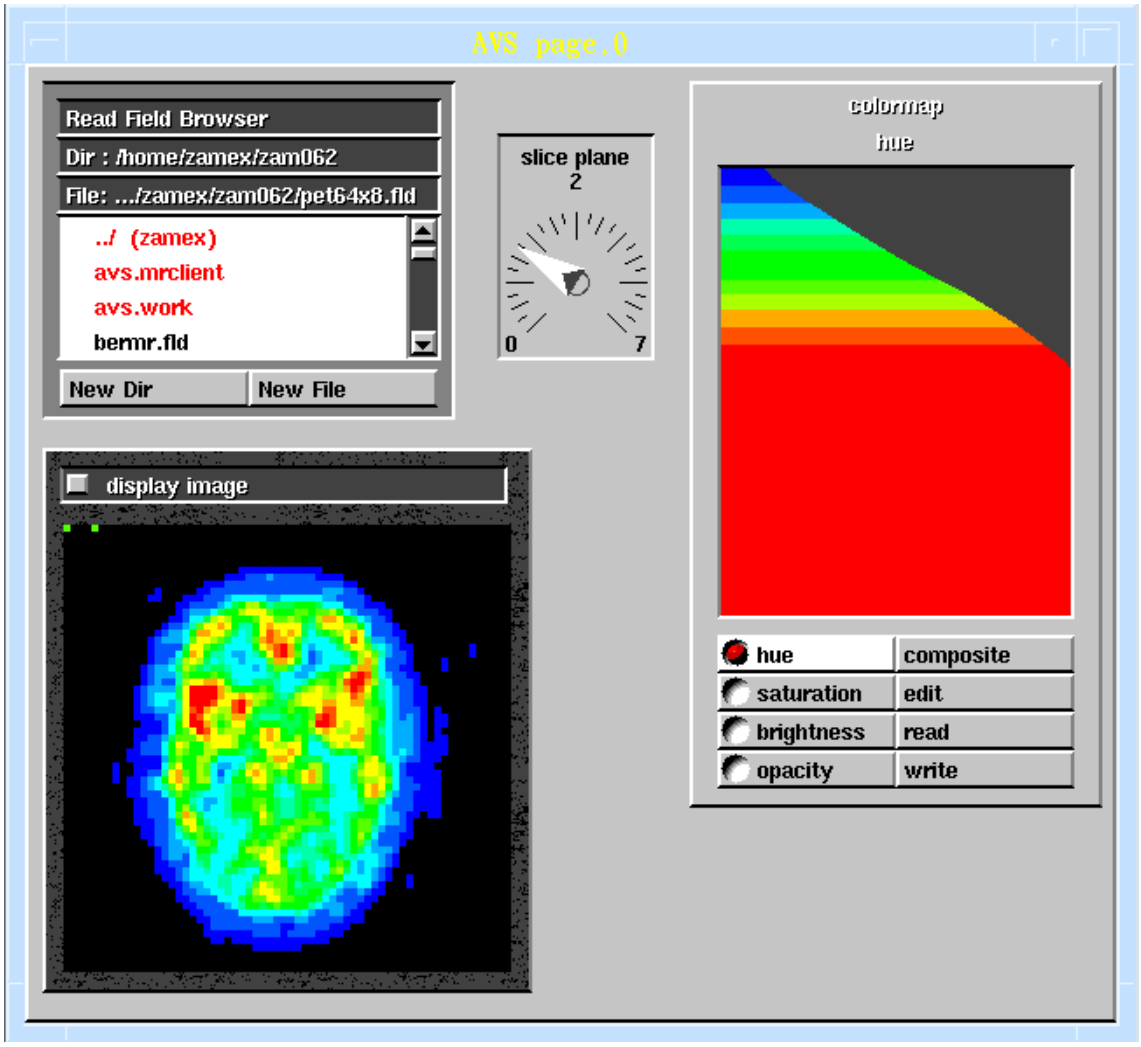


Abbildung 5.2: Bedienungsoberfläche für das Beispielnetzwerk

5.4 Graphische Programmierumgebung

Für die Erstellung von Applikationen unter AVS stehen eine Reihe von Werkzeugen zur Verfügung, wobei der Netzwerk-Editor eine zentrale Rolle einnimmt, siehe Abb. 5.3 . Innerhalb des Netzwerk-Editors lassen sich auf einfache Weise Netzwerke erstellen, die die AVS-Applikation bilden. Aus der Modulbibliothek kann dabei auf eine Reihe von bereits vorhandenen Modulen zurückgegriffen werden, die bereits in Abschnitt 5.2 in die vier Gruppen 'Data Input', 'Filter', 'Mapper' und 'Data Output' unterteilt worden sind. Diese Modulbibliothek [2] findet sich in der oberen rechten Ecke des Netzwerk-Editors wieder.

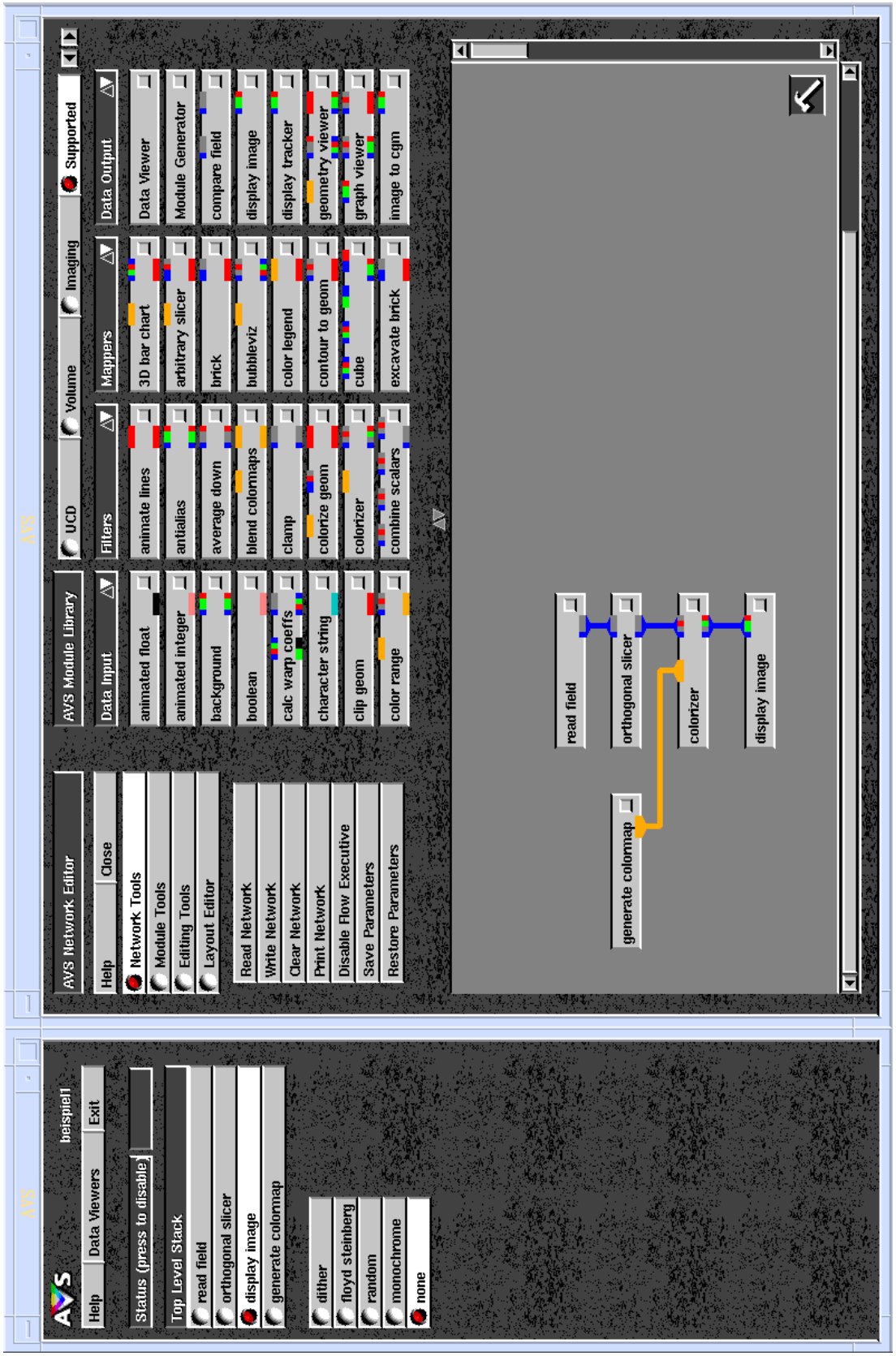


Abbildung 5.3: Graphische Programmierumgebung

Die Module können mit Hilfe der Maus auf die Arbeitsfläche in den unteren Teil des Editors gelegt werden. Ein auf die Arbeitsfläche gebrachtes Modul wird als Prozeß im Rechner gestartet und steht dem Benutzer danach interaktiv zur Verfügung. Indem man ein Modul mit der Maus unter den Hammer in der linken unteren Ecke legt, wird dieses Modul wieder zerstört und der Prozeß aus dem Rechner entfernt. Die Ein- und Ausgänge eines auf der Arbeitsfläche liegenden Moduls lassen sich interaktiv mit anderen Modulen verbinden. Die verschiedenen Verbindungen werden dabei farbig dargestellt. Es lassen sich nur solche Ein- und Ausgänge miteinander verbinden, die eine passende Farbkodierung aufweisen. Farbe und Farbkodierung geben in diesem Zusammenhang Art und Typ der übergebenen Daten an, vgl. [3]. Die möglichen Verbindungen werden graphisch dargestellt, wenn man mit der Maus versucht, zwei Punkte zu verbinden; dabei können nur vorgeschlagene Verbindungen eingegangen werden. In der gleichen Weise, wie sich Verbindungen herstellen lassen, kann man sie auch wieder interaktiv mit Hilfe der Maus entfernen.

Die zu den Modulen gehörenden Bedienungselemente erscheinen in dem linken abgetrennten Fenster. Da nicht alle Elemente aller Module gleichzeitig dargestellt werden können, legt AVS einen Stapel in diesem Fenster an. Drückt man einen Knopf aus dieser Stapelleiste, erscheinen die Bedienungselemente dieses Moduls. Eine weitere Möglichkeit, das Bedienfeld eines Moduls zu erhalten, ergibt sich, wenn man mit der linken Maustaste den kleinen Knopf an der rechten Seite eines Moduls bedient. Benutzt man die rechte Maustaste, so erhält man ein Fenster, in dem sich Eigenschaften eines Moduls ändern lassen, wie z.B. der Name oder die Sichtbarkeit der Ein- oder Ausgänge. Hier ist es auch möglich, die interaktiven Parameter der Module, die eigentlich über das Bedienfeld eingestellt werden, nach außen sichtbar zu machen und als gewöhnliche Moduleingänge zu bedienen, sowie eine Kurzanleitung zu einem Modul aufzurufen, soweit diese vorhanden ist.

Da ein Netzwerk unter Umständen sehr umfangreich und unübersichtlich werden kann, bietet das System die Option, eine Reihe von Modulen zu einem Makro-Modul zusammenzufassen. Die entsprechenden Ein- und Ausgangsdatenpfade werden dabei automatisch generiert. Das Makro-Modul erscheint hierbei auf der Oberfläche wie jedes andere gewöhnliche Modul.

Es ist möglich, erstellte Netzwerke mit ihren Parametern auszudrucken, zu laden und zu speichern. Will man mehrere Parameter der Module gleichzeitig ändern, kann man zu diesem Zweck die Datenflußkontrolle ausschalten. Somit wird bei der Veränderung eines Parameters nicht sofort die gesamte darunterhängende Netzstruktur abgearbeitet.

Für die Erstellung von Benutzeroberflächen stellt der 'Layout Editor' ein sehr nützliches Hilfmittel dar. Er bietet die Möglichkeit, die einzelnen Bedienungselemente in Fenstern zusammenzufassen, um so eine Gesamtoberfläche für den Endanwender zu erzeugen. Diese Arbeit läßt sich wie die übrigen Änderungen interaktiv mit Mausunterstützung durchführen. Das Fenster aus Abb. 5.2 wurde auf diese Weise erstellt.

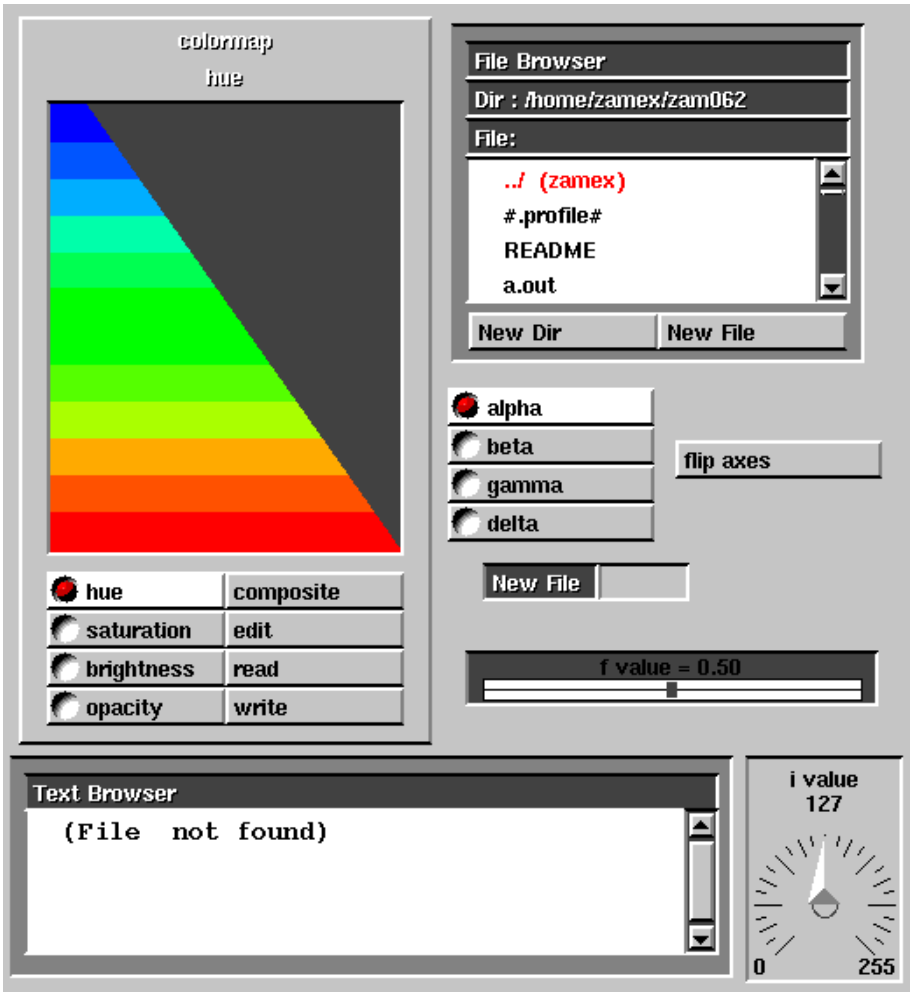


Abbildung 5.4: Vordefinierte Bedienungselemente

5.5 Verwendung eigener Routinen

Neben der in den vorangegangenen Abschnitten beschriebenen Benutzung bereits bestehender Komponenten, bietet AVS die Möglichkeit, eigene Routinen als Module zu konzipieren und in das System zu integrieren. Mit Hilfe des Moduls 'Module Generator' läßt sich das Gerüst, siehe Abb. 5.5, eines Moduls in C-Code erstellen. Dabei ist es möglich, die Definition der Ein- und Ausgänge interaktiv vorzunehmen. Neben den bereits vorgefertigten AVS-Datenstrukturen lassen sich auch eigene Ein- sowie Ausgangsdatenstrukturen definieren, die außerhalb von AVS beschrieben werden müssen. AVS bietet Datenstrukturen für Felddaten, Geometriedaten, die Finite-Elemente-Methode, Molekülstrukturen sowie eine spezielle Datenstruktur für Farbtafeln. In dieser Arbeit steht der Felddatentyp im Vordergrund. Dieser Datentyp setzt sich aus mehreren Teilen zusammen. Neben den Strukturen für das eigentliche Datenfeld sind Elemente vorhanden, die Ausmaße, Aufbau und Lage der Daten beschreiben [1].

Neben den vordefinierten Datentypen existiert eine Reihe von vorgegebenen Bedienungselementen, die für eigene Module benutzt werden können. Die vorgegebenen Elemente werden in gleicher Weise genutzt wie bei bereits existierenden Modulen. Abb. 5.4 zeigt eine Auswahl dieser Bedienungselemente. Neben den bereits in Abschnitt 5.3 angesprochenen, finden sich Elemente wie Schieberegler (*f value*), ein Fenster, um Daten als Klartext zu sehen (*Text Browser*), Umschaltknöpfe, wobei nur einer aktiv sein kann (*alpha, beta, gamma, delta*) oder Drucktaster (*flip axes*) sowie Felder, in die Text eingegeben werden kann (*New File*). Während Knöpfe und Texteingaben Zeichenketten als Parameter übergeben, lassen sich mit den Dreh- und Schiebereglern numerische Eingaben mit numerischen Datentypen erzeugen. Die Namen der Bedienungselemente sind frei wählbar und werden unter diesen Namen im Modultext angesprochen.

In Abb. 5.5 erfolgt nach den einleitenden Kommentaren zum Aufbau des Moduls die Einbindung von AVS-Definitionen sowie benutzereigenen Definitionen mittels *include*. Der folgende Block stellt die Routinen für die Modulbeschreibung dar. Hiermit werden die einzelnen Bedienungselemente, sowie die Ein- und Ausgänge dem AVS bekannt gemacht, wobei es dem Benutzer freisteht, noch weitere Definitionen einzufügen. Der sich anschließende Block beinhaltet das Gerüst für die Benutzerroutine. Der Aufruf der Benutzerfunktion erfolgt wie bei jedem anderen C-Unterprogrammaufruf. Die übergebenen Parameter liegen durch die vorher stattgefundene interaktive Schnittstellendefinition bereits fest und bezeichnen Aus- und Eingänge sowie Parameter des Moduls. Innerhalb dieses Blockes steht es dem Benutzer frei, jede Art von Algorithmus zu schreiben, wobei ihm alle Funktionen des Compilers zur Verfügung stehen. Der Modulgenerator legt auf Wunsch auch ein 'makefile' an, das die Datenabhängigkeit des Moduls von den verschiedensten Bibliotheken angibt. Anschließend ruft man den Compiler aus dem System heraus auf. Es handelt sich hierbei um denselben Compiler wie er einem UNIX-System üblicherweise zur Verfügung steht. Das hier präsentierte Beispiel zeigt ein Gerüst für eine Schwellwertoperation. Aus diesem Grund findet man als aufrufende Parameter neben den Ein- und Ausgangsdatenfeldern auch den Parameter *schwelle* für die Angabe des Schwellwertes. Im folgenden Abschnitt wird die Routine zur Initialisierung des Moduls bereitgestellt. Im letzten Block ist Platz für vom Benutzer geschriebene Unterroutinen.

Das hier vorgestellte Modul ist als 'Subroutine' definiert, damit wird es als Unterroutine in AVS betrachtet und behandelt, d.h. die Kontrolle der Änderung von Datenein- und ausgängen sowie Parametern übernimmt der AVS-Systemkern und spricht ggf. das Modul an. Eine andere Art Module stellen Routinen dar, die als 'Coroutine' definiert sind. Diese Module können unabhängig von AVS agieren, sind damit aber selber für Änderungen verantwortlich.

```
/* Module Name: "level" (Filter)                                        */
/* Author: Norbert Esseling                                             */
/* Date Created: Wed Feb 16 18:56:38 1994                               */
/*                                                                      */
/* This file is automatically generated by the Module Generator (mod_gen)*/
/*                                                                      */
/* input 0 "eingang" field byte REQUIRED                                */
/* output 0 "ausgang" field byte                                        */
/* param 0 "schwelle" idial 128 0 255                                   */
/* End of Module Description Comments                                   */

#include <stdio.h>
#include <avs/avs.h>
#include <avs/port.h>
/* -->START OF USER-SUPPLIED CODE SECTION #1 (INCLUDE FILES, GLOBAL VAR.)*/
/* <-- END OF USER-SUPPLIED CODE SECTION #1                              */

/* ************************************/
/*  Module Description                */
/* ************************************/
int level_desc()
{
        int in_port, out_port, param;
        extern int level_compute();
        AVSset_module_name("level", MODULE_FILTER);

        /* Input/Output Port & Parameter Specifications              */
                  ...
        AVSset_compute_proc(level_compute);
/* -->START OF USER-SUPPLIED CODE SECTION #2 (ADDITIONAL SPECIFICATION INFO)*/
/* <-- END OF USER-SUPPLIED CODE SECTION #2                              */
        return(1);
}

/* ************************************/
/* Module Compute Routine             */
/* ************************************/
int level_compute( eingang, ausgang, schwelle, wert)
        AVSfield_char *eingang;   /* Eingangsdatenfeld */
        AVSfield_char **ausgang; /* Ausgangsdatenfeld */
        int schwelle;           /* Schwelle */
{
/* --> START OF USER-SUPPLIED CODE SECTION #3 (COMPUTE ROUTINE BODY)   */
                  ... Benutzerroutine ...
/* <-- END OF USER-SUPPLIED CODE SECTION #3                              */
        return(1);
}

/* ********************************************************************/
/* Initialization for modules contained in this file.                  */
/* ********************************************************************/
                            ...
AVSinit_modules()
{
        AVSinit_from_module_list(mod_list, NMODS);
}
/* --> START OF USER-SUPPLIED CODE SECTION #4 (FUNCTIONS, UTILITY ROUTINES)*/
/* <-- END OF USER-SUPPLIED CODE SECTION #4                              */
```

Abbildung 5.5: Beispiel für ein Modulgerüst eigener Routinen

Kapitel 6

Sequentieller Ansatz

In diesem Kapitel wird die Implementierung des Kopf-und-Hut-Verfahrens nach der Grundidee von Pelizzari et al. [40] mit Hilfe der Visualisierungs- und Entwicklungsumgebung AVS beschrieben. Ausgehend von dem Algorithmus als Gesamtnetzwerk findet im folgenden eine Aufspaltung bis in die einzelnen Module statt. Im Abschnitt 6.5 sind die Bedienfelder der einzelnen Module gesammelt erklärt.

6.1 Gesamtübersicht der sequentiellen Implementierung

Um das Verfahren nach Pelizzari et al. [40] in ein modulares Netzwerk nach AVS-Struktur umzusetzen, muß dieser Anpassungsalgorithmus in seine wesentlichen Bestandteile zerlegt werden. Jeder dieser Bestandteile kann dabei als abgeschlossene Einheit angesehen werden, die durch ihr Verhalten und ihre Schnittstellen nach außen definiert wird. Untersucht man das Kopf-und-Hut-Verfahren näher, so ergeben sich vier Teilprobleme:

- **MRT-Oberflächenerzeugung:** Aus den bestehenden MRT-Daten muß die Oberfläche des Patientenkopfes detektiert und extrahiert werden.
- **PET-Punkteerzeugung:** Eine die Außenkontur des PET-Datensatzes beschreibende Punktmenge ist zu erzeugen.
- **Geometrische Anpassung:** Die beiden erzeugten Oberflächendatensätze sollen zur Deckung gebracht werden.
- **Datenneuanordnung:** Aus den Parametern der Anpassung werden den PET-Datenschichten entsprechende MRT-Datenschichten zugeordnet.

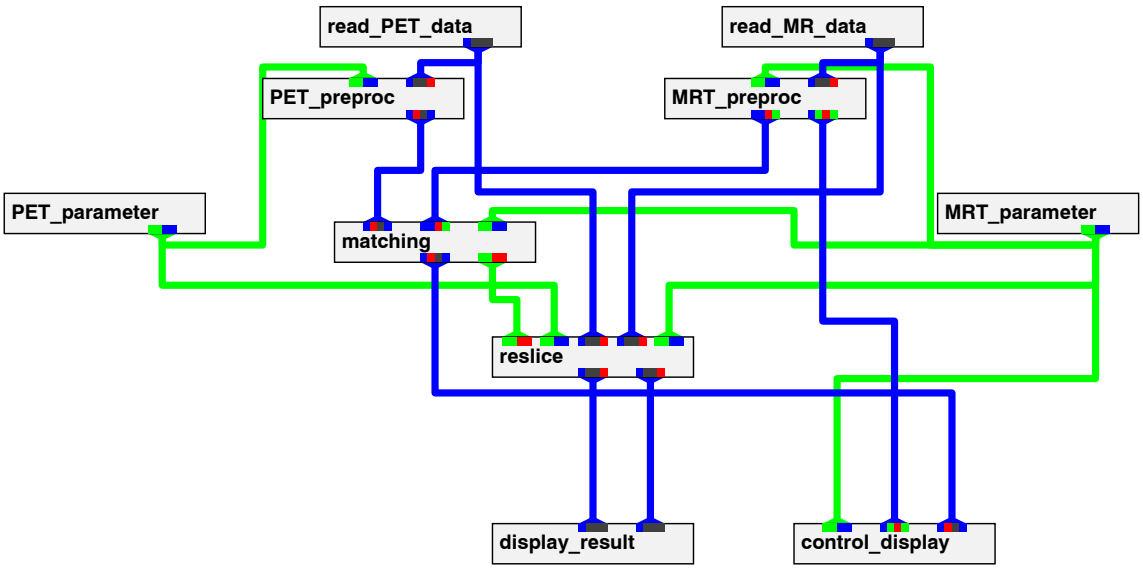


Abbildung 6.1: Gesamtnetzwerk des implementierten sequentiellen Verfahrens

Abb. 6.1 zeigt eine Gesamtnetzübersicht des implementierten Verfahrens. Die Bearbeitung der Teilprobleme findet in den vier Modulen *MRT_preproc* (MRT-Oberflächenerzeugung), *PET_preproc* (PET-Punkteerzeugung), *matching* (Geometrische Anpassung) und *reslice* (Datenneuanordnung) statt. Die beiden Module *MRT_preproc* und *PET_preproc* sind voneinander unabhängige Vorverarbeitungsschritte. Ihre Ausgangsdaten dienen den nachfogenden Einheiten als Eingangsdaten.

Die Module *read_PET_data* und *read_MR_data* übernehmen das Einlesen der PET- und MRT-Daten in das gezeigte Bearbeitungsnetzwerk. Mit Hilfe der Module *PET_parameter* und *MRT_parameter* lassen sich weitere, den einzelnen Datensatz beschreibende Parameter erzeugen, die für die Arbeit einiger Module benötigt werden und gleichzeitig an zentraler Stelle beeinflußbar sind. Die Module *display_result* und *control_display* zeigen das Endergebnis bzw. ein Kontrollbild an, sie stellen den Datenausgang des Netzes dar. Alle bisher genannten Module sind Makro-Module, die im weiteren zerlegt und näher betrachtet werden. Die die Module verbindenden Datenpfade geben den Datenfluß der Module untereinander an.

6.2 Eingangsdaten

Die in dieser Arbeit verwendeten Datensätze bestehen aus einem Schichtdatensatz, der mit Hilfe der Magnet-Resonanz-Tomographie (MRT) erstellt wurde, und einem Schichtdatensatz, der mit Hilfe der Positronen-Emissions-Tomographie (PET) erstellt wurde.

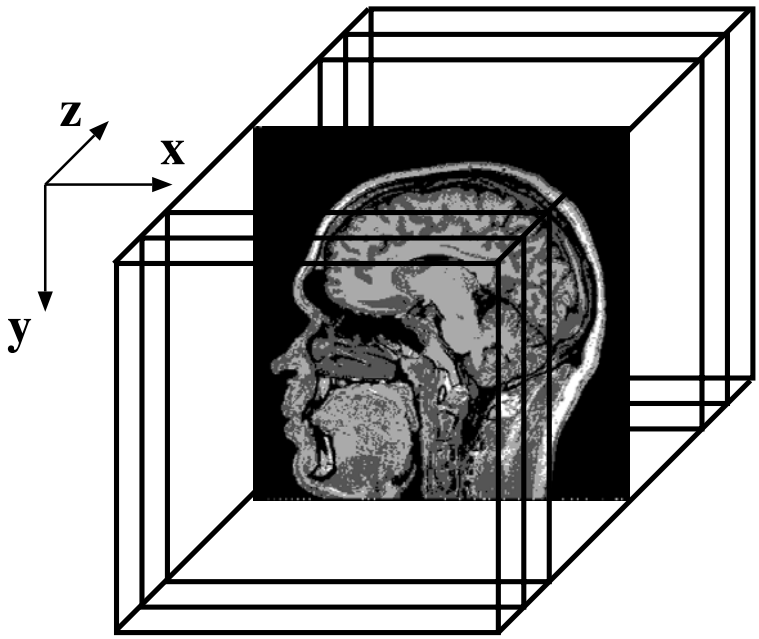


Abbildung 6.2: MRT-Eingangsdaten

Abb. 6.2 zeigt den MRT-Datensatz im Überblick. Er besteht aus 128 Schichtbildern mit je 256 Punkten in x- und y-Richtung. Die Pixelgröße in x- und y-Richtung beträgt jeweils 1 mm, während die Schichtdicke in z-Richtung 1,32 mm beträgt. Wie in der Abbildung zu erkennen ist, liegt eine sagittale Schichtorientierung vor. Die Ursprungsdaten sind in den AVS-Felddatentyp *AVSfield_char* umgewandelt worden, dessen Elemente einzelne Bytes sind. Auf diese Weise lassen sich 256 verschiedene Grauwerte ablegen, die aus dem Originaldatensatz ermittelt werden können. Der Aufbau der Originaldaten findet sich in [8] . Für eine an das Kopf-Hut-Verfahren anschließende Segmentierung ausgewählter Geweberegionen ist es wichtig, eine evtl. vorhandene Normierung innerhalb der einzelnen Schichten untereinander in Einklang zu bringen, bevor man den hier erstellten Algorithmus nutzt.

In Abb. 6.3 ist der verwendete PET-Datensatz als Übersicht gezeigt. Es existieren insgesamt 15 transversale Schichtbilder, die aus je 128 Punkten in x- und y-Richtung bestehen. Während die physikalische Auflösung der Bilder ca. 7 mm beträgt, ist die Kantenlänge eines abgespeicherten Datenpixels in der x- und y-Richtung jeweils 2 mm. Es wird der gleiche AVS-Datentyp wie bei den MRT-Daten verwendet, wobei auch hier eine Konvertierung der Originaldaten nötig ist. Die Schichtdicke ist bei diesen Bildern mit 6,4 mm angegeben.

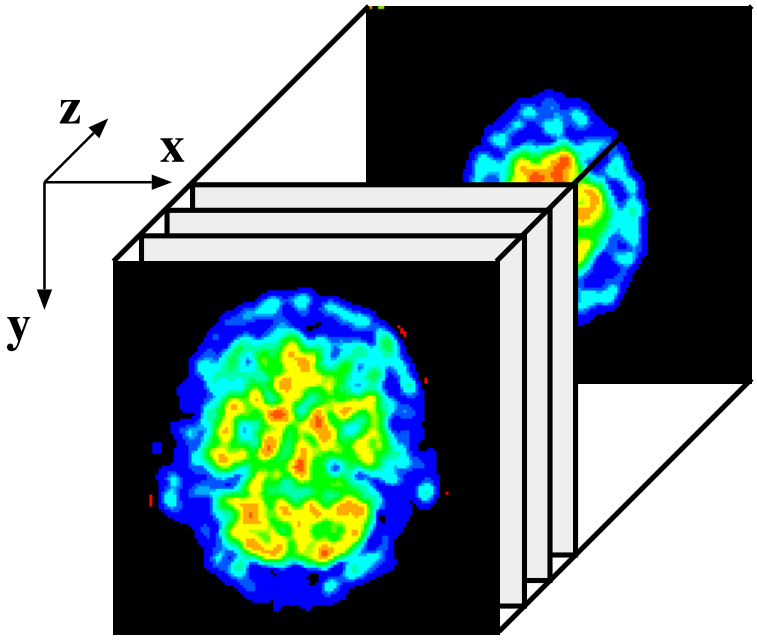


Abbildung 6.3: PET-Eingangsdaten

Die Umwandlung der Ursprungsdaten in den AVS-Datentyp *AVSfield_char*, vgl. [1], wurde außerhalb des AVS vorgenommen. Jedes Originaldatum wurde in einen Wert zwischen 0 und 255 umgewandelt. Die Ablage der Daten erfolgt wie bei C üblich zeilenweise. Mehrere Zeilen hintereinander beschreiben eine Schicht. Diese Schichten werden anschließend nacheinander in eine Datei geschrieben. Diese Daten-Bytes bilden den Datenblock des AVS-Datensatzes. Zum kompletten AVS-Datensatz fehlt nur noch der am Anfang jeder Datei stehende Beschreibungsparameterblock, der als 'Header' bezeichnet wird. Ein Header für eine Datei kann wie folgt aussehen:

```
# AVS field file              (Header)
# creation date: 21.2.'94
#
ndim=3                  # number of dimensions in the field
dim1=128                # dimension of axis 1
dim2=128                # dimension of axis 2
dim3=15                 # dimension of axis 3
nspace=3                # number of physical coordinates per point
veclen=1                # number of components at each point
data=byte               # data type (byte, integer, float, double)
field=uniform           # field type (uniform, rectilinear, irregular)
^L^L>>>>>>>(Data)<<<<<<<<<
```

Während die beiden Control-L-Zeichen am Ende des Headers den Beschreibungsblock beenden und den Anfang der eigentlichen Daten kennzeichnen, werden Kommentare mit einem #-Zeichen abgetrennt. Die Angabe *ndim* bezieht sich auf die

vorhandenen Dimensionen des Datenfeldes, während die Werte *dim1* bis *dim3* die Ausdehnung in der jeweiligen Dimension angeben. Die hier angegebenen Werte sind typisch für einen PET-Datensatz. Der Aufbau für MRT-Datensätze sieht prinzipiell gleich aus, wobei sie sich in der Dimensionierung unterscheiden. *nspace* bezeichnet die Anzahl der physikalisch vorhandenen Dimensionen, in diesem Fall hat sowohl das Datenfeld drei Dimensionen bei der Datenablage, als auch drei Dimensionen bei der Interpretation der Position eines Punktes im Raum. Die Variable *veclen* gibt Auskunft über die Anzahl der Komponenten pro Pixelpunkt. Da hier nur der Grauwert eines Punktes interessant ist, benötigt man lediglich einen Skalar. Aus diesem Grund ist die Anzahl der Komponenten eines Datums 1, was sich hier durch *veclen=1* ausdrückt. Die Daten werden als Bytes abgespeichert, dies wird AVS durch die Angabe *data=byte* mitgeteilt. Als letztes findet sich im Header eine Information über die Art des Feldes. Die Zuweisung *field=uniform* kennzeichnet hierbei einen Feldtyp, dessen Abstände zwischen den Punkten regelmäßig ist, wobei eine rechtwinklige Anordnung der Daten angenommen wird (Quaderstruktur). Es gibt noch eine Reihe von anderen Parametern, die es ermöglichen, einen Datensatz näher zu beschreiben; sie werden hier jedoch nicht benutzt.

An den beiden hier vorgestellten Datensätzen erkennt man deutlich die auftretenden Probleme, wenn man Daten dieser beiden medizinischen Abbildungsverfahren aneinander anpassen will. Neben der verschiedenen Orientierung haben diese Datensätze eine unterschiedliche Auflösung, die bei der Anpassung berücksichtigt werden muß.

Die beschriebenen Schichtdaten werden jeweils durch die Module *read_MRT_data* für die MRT-Daten und *read_PET_data* für die PET-Daten gelesen. Diese beiden Module sind Makro-Module, die aus dem AVS-Modul *read field* bestehen. Die für die Dateiauswahl vorgesehenen Bedienungselemente finden sich in Abschnitt 6.5 .

Mit den beiden Modulen *PET_parameter* und *MRT_parameter* aus Abb. 6.1 wird jeweils zusätzlich ein Satz Daten erzeugt, der die Eingangsdaten näher charakterisiert. Der ausgegebene Datentyp ist der Benutzerdatentyp *param_type* und in C-Syntax wie folgt implementiert worden:

```
typedef struct
       { int   min_x;   /* minimales x */
         int   max_x;   /* maximales x */
         int   min_y;   /* minimales y */
         int   max_y;   /* maximales y */
         int   min_z;   /* minimales z */
         int   max_z;   /* maximales z */
         float xsize;   /* Pixelgroesse in x-Richtung */
         float ysize;   /*       "      in y-Richtung */
         float zsize;   /*       "      in z-Richtung */
       } param_type;    /* Name des Benutzertyps */
```

Während der erste Block die minimal und maximal zu berücksichtigenden Pixel bei der Bearbeitung des Datensatzes angibt, beinhaltet der zweite Block Angaben

über den Abstand eines Datenpunktes zu einem folgenden für alle drei Raumrichtungen. *PET_parameter* und *MRT_parameter* sind Makro-Module, die ein Dateneingabemodul mit dem Namen *machine_param* beinhalten, der die soeben gezeigte Datenstruktur belegt.

6.3 Verarbeitungsalgorithmen

In diesem Abschnitt werden die Algorithmen zu den in Abschnitt 6.1 genannten Teilproblemen beschrieben und näher erläutert.

6.3.1 MRT-Vorverarbeitung

Wie bereits erwähnt, besteht das Bearbeitungsnetzwerk aus zwei Vorverarbeitungszweigen. In diesem Abschnitt wird die MRT-Vorverarbeitung betrachtet. Die Aufgabe besteht hier darin, die Oberflächenstruktur der MRT-Daten zu erzeugen. Bildlich gesprochen, erzeugt die MRT-Vorverarbeitung den 'Kopf' des hier verwendeten Verfahrens der Anpassung.

Abb. 6.4 zeigt das aufgelöste Makro-Modul *MRT_preproc* aus der Gesamtübersicht. Der Block *IN→MRT_preproc* kennzeichnet die Eingangsschnittstelle des Makro-Moduls. Der linke Datenzweig dieses Blockes transportiert die zusätzlichen Datensatzangaben, die aus dem Modul *MRT_parameter* stammen. Auf der rechten Seite werden die eigentlichen MRT-Daten herangeführt, die aus dem Modul *read_MR_data* kommen. An die Ausgabeschnittstelle *OUT→MRT_preproc* werden auf rechten Seite die mit diesem Modul zu erzeugenden Tiefenkarten, vgl. unten, sowie die zu erstellenden Konturlisten übergeben. Die Konturlisten dienen im folgenden nur Kontrollzwecken und sind nicht Bestandteil des eigentlichen Kopf-und-Hut-Verfahrens. Das genaue Aussehen der hier erzeugten Ausgangsdaten wird in den Abschnitten über die Konturgenerierung und die Tiefenkartenerzeugung näher betrachtet.

Schwellwertbildung

Der hier beschriebene Verarbeitungsschritt stellt die erste Operation der MRT-Verarbeitungskette dar. Für die Generierung einer Kontur bzw. einer Oberfläche, die ein Volumen beschreibt, sind nur die Grenzen eines Volumens interessant und nicht der genaue Grau- oder Farbwert an jeder Stelle des Datensatzes. In diesem Fall reicht die Information, ob ein Datenpunkt zum Objekt oder zu dem umgebenden Hintergrund gehört. Hier ist das Unterscheidungskriterium zwischen Hintergrund und Objekt, ob die Intensität des Grauwertes eine bestimmte Schwelle überschreitet. Ist dies der Fall, klassifiziert man diesen Datenpunkt als zum Objekt gehörig. Diese Operation wird in der digitalen Bildverarbeitung als 'Schwellwertoperation' bezeichnet und ist in Abschnitt 4.1.1 näher erläutert.

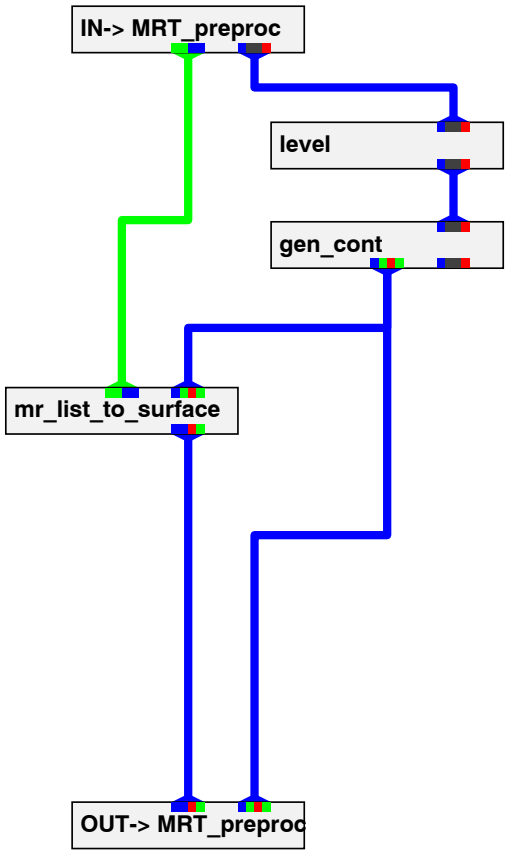


Abbildung 6.4: Netzwerk für die MRT-Vorverarbeitung

Abb. 6.5 zeigt den Effekt dieser Operation auf ein MRT-Grauwertbild. Die linke Seite zeigt das Eingangsbild, während auf der rechten Seite das zweiwertige Ausgangsbild zu sehen ist. Das Ausgangsbild enthält nur die beiden Werte 0 und a. Für die soeben beschriebene Operation wurde das Modul *level* aus Abb. 6.4 erstellt.

Der Aufruf der Modulberechnungsroutine ist definiert durch:

```
int level_compute( eingang, ausgang, level, to)
        AVSfield_char *eingang;   /* Eingangsdatenfeld */
        AVSfield_char **ausgang; /* Ausgangsdatenfeld */
        int level;               /* Schwelle */
        int to;                 /* Ausgangswerte bei x>Schwelle */
{ .. }
```

Der Modulparameter *to* entspricht dem Wert a und *level* dem gewünschten Schwellwert. Der Wert für a ist hier fest auf den Wert 255 eingestellt und darf nicht geändert werden, da ansonsten folgende Module aufgrund ihrer Implementierung nicht mehr korrekt arbeiten können. Aus diesem Grunde ist nur der Schwellwertparameter in die Bedienungsoberfläche integriert, siehe Abschnitt 6.5 .

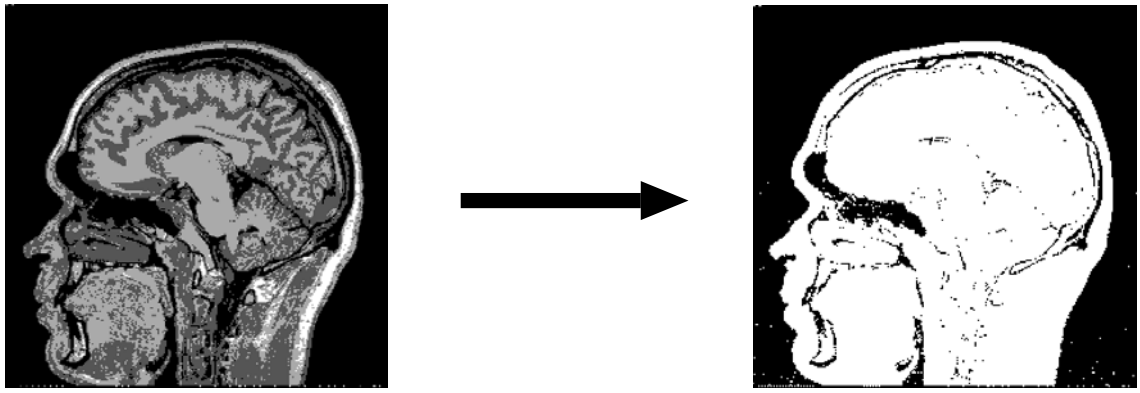


Abbildung 6.5: Schwellwertbildung

Die Ein- und Ausgangsdaten werden als Felder des AVS-Datentyps *AVSfield_char* angenommen bzw. übergeben. Neben den eigentlichen Daten wird Variablen diesen Datentyps auch die Dimensionierung des dreidimensionalen Datenfeldes übergeben. Das Modul übernimmt die Daten aus dem rechten Datenzweig der Makro-Schnittstelle, während es die Daten an das im folgenden beschriebene Modul *gen_cont* weitergibt.

Konturgenerierung

Die Aufgabe des in diesem Abschnitt beschriebenen Moduls *gen_cont* aus Abb. 6.4 ist es, geschlossene Konturen in den im vorhergehenden Abschnitt erzeugten Binärbildern zu finden. Diese Konturen werden separat für jedes eingehende Schichtbild erzeugt und als sog. Konturlisten, vgl. Abschnitt 4.1.2, abgelegt. Hierbei wird die ursprüngliche Bildinformation auf eine Reihe von Kettenkodelisten reduziert.

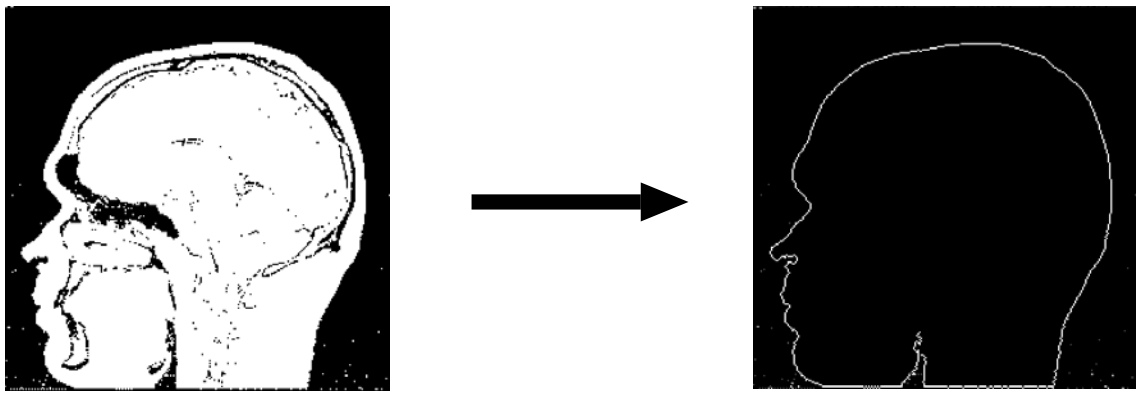


Abbildung 6.6: Konturgenerierung

Abb. 6.6 zeigt links eines der erzeugten Binärschichtbilder, während die andere Seite das Bild der zum Eingangsbild gehörenden Konturen zeigt. Dieses Bild kann als Kontrollbild dem nicht weiter verbundenen Ausgang entnommen werden, wobei der

Datentyp dem Eingangsdatentyp entspricht (*AVSfield_char*). Innerhalb dieses Datensatzes sind die Anfangspunkte einer Kontur mit dem Wert 63 belegt, während übrige Konturpunkte mit 127 belegt werden. Konturpunkte können nur diejenigen Punkte werden, die im Eingangsbild den Wert 255 aufweisen, siehe Schwellwertbildung im vorhergehenden Abschnitt.

Der Algorithmus zur Konturfindung startet seine Suche nacheinander von allen vier Bildseiten. Wird von der linken Seite gesucht, so wird jede Zeile von links nach rechts nach einem möglichen Konturpunkt durchsucht. Die Suche wird sofort abgebrochen, wenn ein bereits markierter Konturpunkt gefunden wird. Findet der Algorithmus einen noch nicht erkannten Konturpunkt, so setzt zunächst eine Konturverfolgung im Uhrzeigersinn ein, der sich eine Konturverfolgung in Gegenuhrzeigerrichtung anschließt, die vom gleichen Anfangspunkt ausgeht. Die Verfolgung wird jeweils abgebrochen, wenn zu einem markierten Konturpunkt kein weiterer unmarkierter Folgepunkt bestimmt werden kann oder ein möglicher Kandidat bereits ein markierter Konturpunkt ist. Hiernach ist die Suche in der Zeile, in der der Anfangspunkt der Kontur liegt, beendet.

Für die Folgepunkte innerhalb einer Konturverfolgung kommen nur Punkte in Frage, die in der Achter-Nachbarschaft des gerade aktuellen Konturpunktes liegen. Eine Ausnahme bildet der Punkt, der in Gegenrichtung der alten Fortschreitungsrichtung liegt. Im Normalfall ist dies ein Vorgängerpunkt. In der Regel kommen als Nachfolgepunkte eine Reihe von Punkten in Frage, wobei dann derjenige Punkt bevorzugt wird, der die Außenkontur markiert, also möglichst links gegenüber der alten Suchrichtung liegt, wenn die Kontur im Uhrzeigersinn umfahren wird. Ist die Konturverfolgung beendet, wird versucht in der nächsten Zeile eine neue unmarkierte Kontur zu finden.

Sind alle Konturen von der linken Seite her gefunden, startet der Algorithmus seine Suche von der rechten Seite, anschließend wird von oben und unten nach noch nicht entdeckten Konturen gesucht. Auf diese Weise ist gewährleistet, daß auch Teilkonturen, die zwar von außen direkt erreichbar sind, aber nicht durch Umfahren der Kontur beschreibbar sind, detektiert werden. Auf diese Weise erhält man die Konturen der umfahrbaren Gebiete, wobei nicht jedes Gebiet notwendigerweise in einem Zug umfahren wird.

Die Daten werden als Liste in einem Feld abgelegt, wobei die erste Stelle die Länge der Kontur angibt, während die beiden folgenden Felder die Anfangskoordinaten des ersten Konturpunktes enthalten. Daran schließen sich Richtungskodes im Uhrzeigersinn an, die mit dem Umkehrzeichen (9) beendet werden. Als letztes werden die Richtungskodes, ausgehend vom Anfangspunkt, in Gegenuhrzeigerrichtung gespeichert und mit dem Endezeichen (10) abgeschlossen. Für das Beispiel in Abb. 4.1 ergibt sich demnach die Zahlenkette: 18,2,3,1,2,2,...,7,6,9,10 .

Der Aufruf der Berechnungsroutine von *gen_cont* stellt sich wie folgt dar:

```
int gen_cont_compute( in, outimage, outlist)
        AVSfield_char *in;        /*Eingangsdaten*/
        AVSfield_char **outimage; /*Ausgangskontrollfeld, Schichtbilder*/
        AVSfield_int **outlist;  /*Ausgangsliste*/
{ ... }
```

Während die Eingangsdaten und das Ausgangskontrollfeld als dreidimensionale Felder des Types *AVSfield_char* abgelegt und übergeben werden, handelt es sich bei dem Ausgangslistenfeld um ein Feld, das zweidimensional interpretiert wird. Hieraus folgt, daß die erste Beschreibungskomponente die maximal mögliche Gesamtdatenlänge der Listen innerhalb einer Schicht angibt, während die zweite Komponente die Anzahl der Schichten widerspiegelt. Die dritte Felddimensionierungskomponente erhält hierbei den Wert 1. Für die Konturlisten wurde der Datentyp *AVSfield_int* verwendet, da evtl. Werte größer als 255 auftreten können, wie es z.B. bei der Länge einer Kontur der Fall sein kann. Es wurde auf Benutzung einer sonst üblichen Listenstruktur zugunsten einer Feldstruktur verzichtet, um die Einbettung in das AVS-System handhabbarer zu gestalten. Das Modul besitzt keine Modulparameter und damit auch keine Bedienungselemente.

Tiefenkartenerzeugung

Die mit dem Modul *gen_cont* erzeugten Konturlisten sind für einen schnellen Zugriff auf die örtliche Lage eines Pixel unpraktisch, da nur die Relation von einem Punkt zum nächsten eingetragen wird. Auf der anderen Seite ergibt sich durch die Konturlisten die Möglichkeit, nur solche Konturen zu berücksichtigen, die eine hinreichende Konturlänge aufweisen. Auf diese Weise können Punktstörungen und Störungen, die kleine Gebiete umfassen, unterdrückt werden. Der Wunsch nach einem schnellen Zugriff und die Unterdrückung von Störungen definieren damit die Aufgabe der Tiefenkartenerzeugung.

Eine Tiefenkarte enthält als Information die Abstände der Punkte einer Oberfläche von einer vorher definierten Ebene. Zur Erzeugung der beiden Tiefenkarten für die z-Richtung wird in dem zu betrachtenden Datenraum eine Ebene senkrecht zur z-Richtung bestimmt, die den Raum in zwei gleich große Datenräume zerlegt. Denkt man sich die Ebene bei z=0, so ist dies die Unterteilung in die beiden Teilräume für die positiven und negativen Werte von z. In die bereits erwähnten Tiefenkarten läßt sich nun der Abstand der Konturpunkte von der z-Ebene eintragen. Die Dimensionierung der beiden Karten entspricht dem Maximum der gewünschten Punkte in x- und y-Richtung. Zur Vereinfachung des Datensatzes wird die Dimensionierung einer Schicht im gesamten Datensatz verwendet. Neben den Karten für die z-Richtung existieren weitere vier Karten für die x- und y-Richtung, die entsprechenden Aufbau aufweisen. Abb. 6.7 zeigt auf der rechten Seiten die Tiefenkarte für die negative x-Richtung, die für die frontale Gesichtsregion zuständig ist.

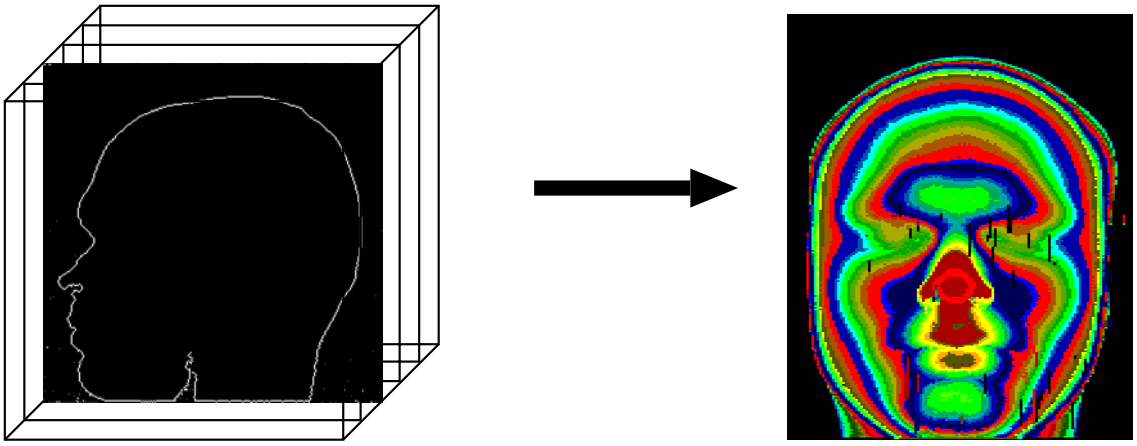


Abbildung 6.7: Tiefenkartenerzeugung

Durch die Erzeugung der Karten wird gleichzeitig der Nullpunkt der Bezugsdaten festgelegt. In allen drei Richtungen wird dabei der gewünschte maximale Schichtindex zum minimalen Wert addiert und durch 2 ganzzahlig dividiert. Diese Extremwerte (*min_x, max_x,ldots*) erhält das Modul aus dem Modul *MRT_parameter* über die Makro-Modul-Schnittstelle.

Wie bereits erwähnt, werden nur solche Konturen für die Tiefenkarten berücksichtigt, deren Länge ein gewisses Maß überschreitet. Für diesen Zweck sind zwei Parameter vorgesehen, wobei der erste eine absolute Länge vorgibt, die erreicht werden muß. Der zweite Parameter gibt die relative Minimallänge innerhalb einer Schicht an, bezogen auf die längste Kontur der Schicht. Diese Parameter finden sich auch in dem Modulaufruf wieder:

```
int mr_list_to_surface_compute( in, machine_para, outpoints,
min_length, min_perc_in_slice)
        AVSfield_int *in;            /* Konturlisten*/
        param_type *machine_para;   /* Maschinenparameter */
        AVSfield_int **outpoints;  /* Ausgangskarten */
        int min_length;           /* Minimale Konturlaenge */
        float *min_perc_in_slice;/* Minimale Laenge relativ zur */
                                /* maximalen Laenge in einer Schicht */
{  ...  }
```

Während Ein- und Ausgangsdaten als *AVSfield_int* übergeben werden, findet der Benutzerdatentyp *param_type* für die benötigten Maschinenparameter Verwendung. Die letzten beiden Aufrufparameter der Routine geben die Parameter für die Bestimmung der minimalen Längen der Konturen an. Die Bedienungselemente für diese Modulparameter sind in die Bedienungsoberfläche aus Abschnitt 6.5 integriert.

6.3.2 PET-Vorverarbeitung

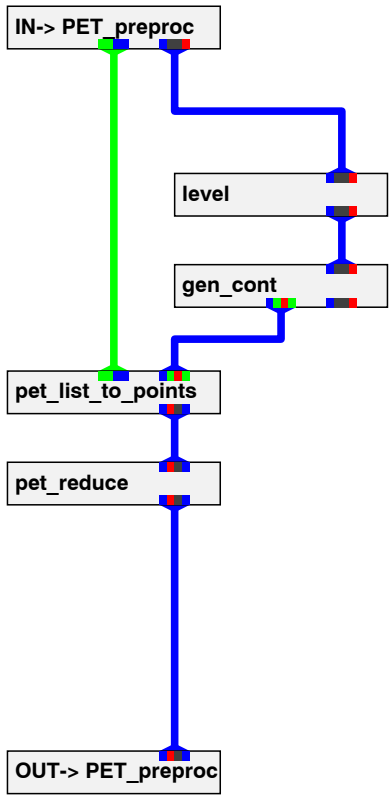


Abbildung 6.8: Netzwerk für die PET-Vorverarbeitung

In diesem Abschnitt wird die PET-Vorverarbeitung beschrieben, wobei einige Module aus der MRT-Vorverarbeitung wiederverwendet werden. Einen Überblick über die PET-Vorverarbeitung zeigt Abb. 6.8, wo das Makro-Modul *PET_preproc* aus Abb. 6.1 weiter aufgelöst wird. Die Aufgabe dieser Vorverarbeitungskette für die PET-Schichtdaten ist die Erzeugung einer Liste von Punkten, die die Ausmaße des Kopfes eines Patienten im PET-Datensatz kennzeichnen und beschreiben. Der linke Datenpfad der Eingangsschnittstelle *IN*→*PET_preproc* transportiert die Datensatzbeschreibungsparameter, wie Ausdehnung und Größe eines Datums. Der verbleibende Datenpfad überträgt die eigentlichen PET-Daten.

Die beiden Module *level* und *gen_cont* werden hier nicht wiederholt betrachtet, da sie identisch mit den gleichnamigen Modulen aus der MRT-Vorverarbeitungskette sind. Das Modul *pet_list_to_points* extrahiert aus den Konturen Punkte, deren Position durch das Modul *pet_reduce* korrigiert werden. An der Ausgangsschnittstelle *OUT*→*PET_preproc* steht die erzeugte Punkteliste der weiteren Verwendung zur Verfügung.

PET-Punkteerzeugung

Ausgehend von den Konturlisten, die von dem Modul *gen_cont* für die PET-Daten erzeugt wurden, werden von dem Modul *pet_list_to_points* Punkte ausgewählt und in einer Liste abgelegt. Bei der Bestimmung der Punkte wird versucht, diese gleichmäßig über die Außenkontur des betrachteten Datensatzes zu verteilen. Hierbei erweist sich die Dateneingabe in Form von Konturlisten als sehr hilfreich, da die Länge einer Liste zu der Länge einer Außenkontur korrespondiert. In dem hier beschriebenen Algorithmus wird deshalb die Gesamtlänge der Außenkontur ermittelt und eine gewünschte Anzahl an Punkten darauf gleichmäßig verteilt.

Wie bei der Tiefenkartenerzeugung in Abschnitt 6.3.1 werden bei diesem Verfahren nur Konturen berücksichtigt, die eine minimale absolute sowie minimale relative Länge innerhalb einer Schicht überschreiten. Aus diesem Grunde existieren hier ebenso die beiden Parameter, die die minimale Absolutlänge und eine Relativlänge in einer Schicht angeben. Ein weiterer Parameter ist die Zahl der zu erzeugenden Punkte. Sollten mehr Punkte gewünscht werden als Konturpunkte vorhanden sind, so wird die Anzahl auf die Zahl der Konturpunkte reduziert. Für die Einordnung der Punkte in ein Koordinatensystem sind die zusätzlichen Angaben (Maschinenparameter) zum Datensatz von entscheidender Bedeutung. Sie ermöglichen die Einordnung der Punkte in ein für beide Datensätze gültiges Koordinatensystem. Der Nullpunkt dieses Systems befindet sich in der Mitte des Datensatzes bzgl. jeder Richtung. Als gemeinsames Bezugssystem wird das System der MRT-Daten (MRT-Welt) verwendet, so daß hier eine Umsetzung der PET-Positionen in die MRT-Welt stattfindet.

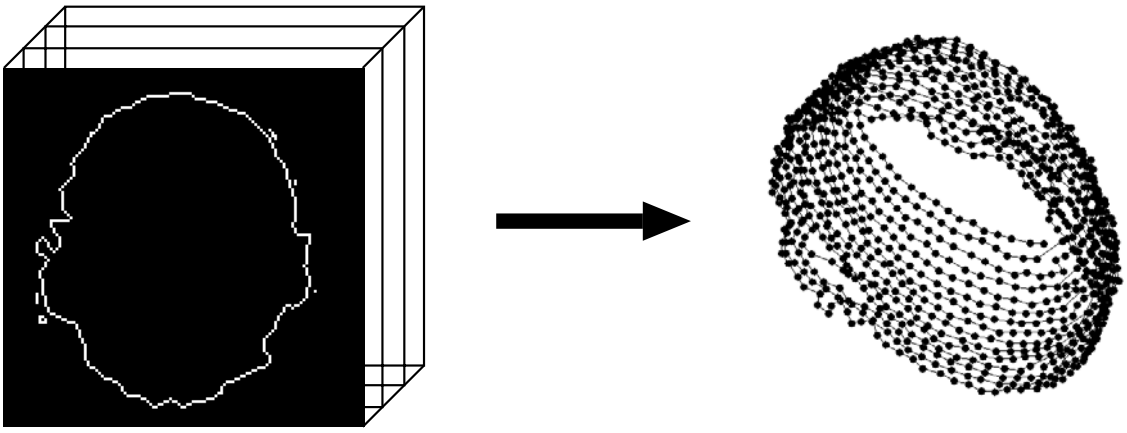


Abbildung 6.9: PET-Punkteerzeugung

Abb. 6.9 zeigt im Überblick die Wirkung dieses Moduls. Als Eingangsdaten finden die durch das Bild auf der linken Seite dargestellten Konturlisten aus dem Modul *gen_cont* Verwendung. Die rechte Seite stellt eine Visualisierung der Ausgangsliste dar. Hierbei sind die Datenpunkte durch kleine Kugeln angedeutet, die zur besseren Orientierung in der Reihenfolge, in der sie erzeugt wurden, verbunden sind.

Der Aufruf der Berechnungsroutine ist definiert durch:

```
int pet_list_to_points_compute( in, machine_para, outpoints,
                        min_length, min_perc_in_slice, num_of_points)
        AVSfield_int *in;                  /* Eingabefeld */
        struct param_type *machine_para; /* Maschinenparameter PET */
        AVSfield_float **outpoints;     /* Feld der Ausgabepunkte */
        int min_length;                /* min. Laenge einer Kontur */
        float *min_perc_in_slice;     /* min. relative Laenge */
        int num_of_points;           /* gewuenschte Anzahl Punkte */
{ ... }
```

In dem Aufruf sind zunächst die Eingabedatenströme definiert, wobei das Eingangsfeld die Konturlisten beinhaltet, wie sie bereits im Abschnitt über die Konturgenerierung erläutert wurden. Wie oben gesagt, werden die zusätzlichen Angaben zum PET-Datensatz benötigt, die hier über den Benutzerdatentyp *param_type* zugeführt werden. Im Ausgangsfeld werden die Positionen der ermittelten Punkte abgelegt, wobei zu den einzelnen Punkten keine weitere Information enthalten ist. Aus diesem Grund existiert kein eigentliches Datenfeld wie bisher, sondern ein Feld, das die Punktpositionen aufnimmt, wie es innerhalb des AVS-Systems üblich ist, siehe [1]. Im Anschluß finden sich die bereits beschriebenen Parameter für den Algorithmus. Die nötigen Bedienungselemente hierfür sind in der Bedienoberfläche integriert.

PET-Auflösungskompensation

Bisher ist die Außenkontur des PET-Datensatzes als Außenkontur der Daten und damit als Kontur des Kopfes interpretiert worden. Dies ist jedoch nur bei einer sehr hohen Auflösung der Fall. Im Normalfall ist die Auflösung der Daten nicht ausreichend für diese vereinfachende Annahme, d.h. eine Punktquelle erscheint im Bild als Kreis mit einem gewissen Radius. Für eine Kontur bedeutet dies, daß die Außenkante der detektierten Kontur weiter außen liegt, als die eigentliche Kopfkontur.

Oft ist es zwar möglich, durch eine Heraufsetzung des Schwellwertes für die Binärbilderzeugung die Kontur auf das eigentliche Maß zu verkleinern, jedoch ergibt sich dann häufig eine Kontur die Einbuchtungen zeigt. Diese Einbuchtungen geben jedoch die Außenkontur nur schlecht wieder. Aus diesem Grunde ist es sinnvoller, den Schwellwert sehr niedrig anzusetzen und die so erhaltene Kontur innerhalb einer Schicht in Richtung Nullpunkt soweit zu schrumpfen, daß die so erzeugte Kontur der tatsächlichen Kopfkontur möglichst gut entspricht. Der nötige Betrag um den die Kontur geschrumpft werden kann, wird hierbei empirisch ermittelt, wobei Messungen der Auflösung als Anhalt dienen. Wie bereits erwähnt, kann man sich die Außenkontur um einen Wert r nach außen versetzt denken. Das Modul *pet_reduce* setzt jeden Punkt der Kontur um den Betrag r nach innen in Richtung Mittelpunkt innerhalb der Schicht, um den soeben beschriebenen Effekt auszugleichen.

Der Aufruf der Berechnungroutine ist:

```
int pet_reduce_compute( inpoints, outpoints, reduce)
        AVSfield_float *inpoints;   /* Eingabefeld */
        AVSfield_float **outpoints;/* Ausgabefeld */
        float *reduce;            /* Wert f. Ausloesungkompensation */
{ ... }
```

Ein- und Ausgabefeld haben hierbei die gleiche Struktur wie die Ausgangsdaten des Moduls *pet_list_to_points*. Der Parameter *reduce* kennzeichnet den oben erwähnten Wert r für die Auflösungskompensation. Ist dieser Parameter größer als 0, so wird die Kontur geschrumpft, andernfalls aufgebläht. Das zugehörige Bedienungselement findet sich in Abb. 6.18 .

6.3.3 Anpassung

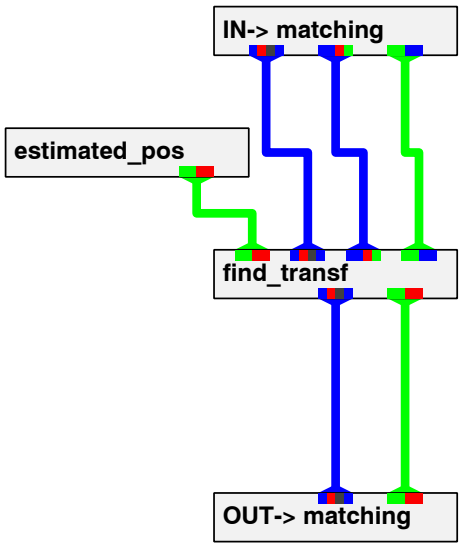


Abbildung 6.10: Netzwerk für die Berechnung der Anpassungsparameter

Nachdem die Vorverarbeitung der Daten beendet ist, kann die eigentliche Berechnung für die Anpassung der Daten stattfinden. In Abb. 6.10 wird das Modul *matching* aus Abb. 6.1 weiter aufgelöst. Die Eingangschnittstelle *IN→matching* besitzt drei Eingangsdatenpfade. Auf der rechten Seite werden die in der PET-Vorverarbeitung ermittelten Punkte übergeben, während der mittlere Pfad die

MRT-Tiefenkarten überträgt. Der verbleibende Datenpfad übernimmt die MRT-Maschinenparameter, damit die Punkte und Entfernungen in den MRT-Tiefenkarten in das MRT-Koordinatensystem eingeordnet werden können. Für die Kontrollausgabe stellt der rechte Ausgabekanal eine nach Ablauf des Algorithmus verschobene und gedrehte Version der PET-Punkte bereit. Der verbleibende Ausgang überträgt die Anpassungsparameter, wie sie in Abschnitt 4.1.3 für eine geometrische Anpassung definiert worden sind. Für diese Parameter existiert der hierfür definierte benutzereigene Datentyp *transf_type*:

```
typedef struct
      { float phi_x;    /* Drehwinkel um x-Achse */
        float phi_y;    /* Drehwinkel um y-Achse */
        float phi_z;    /* Drehwinkel um z-Achse */
        float shift_x;  /* Verschiebung in x-Richtung */
        float shift_y;  /* Verschiebung in y-Richtung */
        float shift_z;  /* Verschiebung in z-Richtung */
        float scale_x;  /* Vergroesserung in x-Richtung */
        float scale_y;  /* Vergroesserung in y-Richtung */
        float scale_z;  /* Vergroesserung in z-Richtung */
      } transf_type;
```

Zunächst sind die Parameter für die Drehung um die einzelnen Achsen definiert. Der hierauf folgende Block enthält die Parameter für die nötige Verschiebung. Als letztes sind hier auch Vergrößerungsfaktoren in die Struktur aufgenommen, die für evtl. Erweiterungen gedacht sind. In dieser Arbeit werden sie nicht genutzt, da es bereits verläßliche Angaben über die Ausdehnung der Datensätze gibt. Da die erstellten Module sie bereits berücksichtigen, werden die Vergrößerungsfaktoren mit 1 vorbesetzt.

Startwertmodul

Für die Anpassung der Daten findet eine Optimierung des Abstandes der Oberflächen der beiden Datensätze statt, vgl. Abschnitt 6.3.3 . Für diese Optimierung muß ein Startwert vorgegeben werden, da der Minimierungsalgorithmus nicht von beliebiger Stelle aus die optimalen Parameter bestimmen kann. Mit Hilfe des Moduls *estimated_pos* ist es dem Benutzer möglich, diesen Anfangswert als Parametersatz zu übergeben. Das Modul bietet an seinem Ausgang die Datenstruktur *transf_type* an, wie sie bereits im vorhergehenden Abschnitt beschrieben ist. Zu dem Modul existiert ein Bedienfeld, vgl. Abschnitt 6.5, auf dem alle Startwerte per Drehzeiger einstellbar sind.

Transformationsfindung

Das Modul *find_transf* aus Abb. 6.10 übernimmt die Aufgabe, die Translations- und Rotationsparameter für eine folgende geometrische Anpassung zu erzeugen. Hierzu benötigt das Modul Informationen über die beiden Datensätze, die zur Deckung gebracht werden sollen.

Die Informationen über die MRT-Oberfläche werden in Form von den in Abschnitt 6.3.1 beschriebenen Tiefenkarten sowie den zusätzlichen Datensatzparametern übergeben. Die Information über die Gestalt der PET-Oberfläche wird in Form der PET-Punktelisten übertragen, vgl. Abschnitt 6.3.2 . Die MRT-Tiefenkarten sind hier als räumlich fixiert zu betrachten, während die PET-Punkte mit Hilfe der aus Abschnitt 4.1.3 bekannten Transformationen beliebig im Raum verschoben und gedreht werden können. Kriterium für eine Anpassung der Daten ist eine möglichst gute Übereinstimmung der erzeugen Oberflächen. Somit handelt es sich hier um eine Optimierungsaufgabe, in der eine Oberfläche in eine andere eingebettet werden soll.

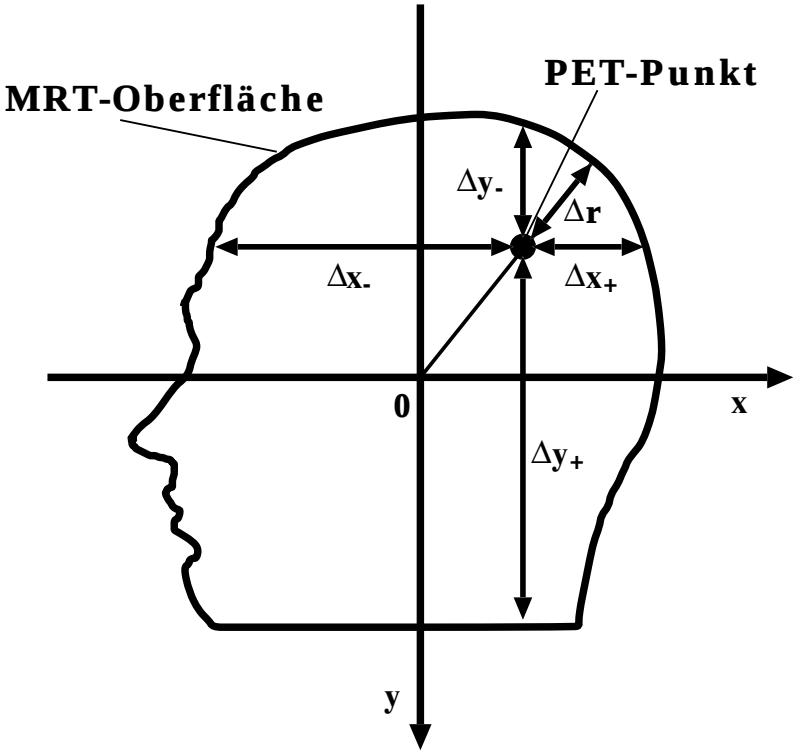


Abbildung 6.11: Bestimmung des Abstands eines PET-Punktes zur MRT-Oberfläche

Wie bereits in Abschnitt 4.2.1 beschrieben, benötigt man für eine Optimierung eine Zielfunktion, deren Minimum sich bestimmen läßt. In diesem Fall wird die Entfernung jedes PET-Punktes zur MRT-Oberfläche gemessen, quadriert und anschließend aufsummiert. Die Bestimmung der Entfernung eines PET-Punktes zur MRT-Oberfläche ist in Abb. 6.11 beispielhaft für den zweidimensionalen Fall dargestellt.

Ausgehend vom PET-Punkt wird in den zwei Raumrichtungen die Entfernung zur MRT-Oberfläche ermittelt. Der Abstand der Kontur von den jeweiligen Symmetrieebenen ist in den Tiefenkarten abgelegt. Will man nun den Abstand eines Punktes von der Oberfläche wissen, so wird die Differenz des in der Tiefenkarte vorhandenen Wertes und der Koordinate des Punktes in der zur betrachtenden Richtung ermittelt. Hieraus ergeben sich Differenzen Δx_-, Δx_+, Δy_- und Δy_+ in dem hier betrachteten Fall. Sind diese Werte bestimmt worden, wird das Minimum der Differenzen als Abstand definiert.

Eine andere Abstandsmessung bestünde darin, die Differenz Δr in radialer Richtung zu bestimmen. Diese Methode ist jedoch datentechnisch ungünstig, da der Schnittpunkt einer Gerade durch Nullpunkt und PET-Punkt mit der Oberfläche zu bestimmen wäre. Die Ermittlung des Schnittpunktes ist dabei nicht analytisch möglich, da die Oberflächen nicht analytisch vorliegen. Die Bestimmung eines Schnittpunktes in Achsenrichtung ist dagegen denkbar einfach, da für jede Achsenrichtung die Entfernung von der zur Achsenrichtung senkrechten Ebene durch den Nullpunkt unmittelbar angegeben ist. Das hier verwendete Verfahren der Verwendung von Tiefenkarten stellt somit eine Näherung zu dem in der Literatur beschriebenen Verfahren [40] dar.

Neben den hier gezeigten vier Richtungen werden zwei weitere in Betracht gezogen und somit das Verfahren auf den dreidimensionalen Anwendungsfall erweitert. Für alle vorher erzeugten PET-Punkte wird auf diese Weise der Abstand zur MRT-Oberfläche bestimmt, die Werte werden quadriert und aufsummiert. Die so erhaltene Größe bildet das Maß für die erreichte Anpassung und ist somit die zu minimierende Größe. Sie entspricht damit der Funktion f aus Abschnitt 4.2.1, wobei die Parameter der Funktion die Verschiebungsparameter im Raum sowie die Drehwinkel um die drei Raumachsen sind.

Für die Minimierung wird ein rechnergestütztes Verfahren genutzt, wobei der Startpunkt aus dem Modul *estimated_pos* übergeben wird. Für diese iterative Minimierung muß an jedem Iterationspunkt die neue Fortschreitungsrichtung bestimmt werden. Abschnitt 4.2.2 enthält hierfür zwei grundlegende Verfahren, die für dieses Modul leicht geändert genutzt wurden:

- **Achsenparalleles Verfahren:** Das Wesen dieses Verfahrens ist die Änderung nur einer Komponente und das Fortschreiten in diese Richtung falls sich der Funktionswert weiter reduziert. Dieses Verfahren wurde hier derart geändert, daß für alle Parameterachsen der Funktionswert für eine Änderung in negativer und positiver Richtung berechnet wird. Anschließend wird als Suchrichtung diejenige Richtung bestimmt, in der der kleinste Funktionwert ermittelt werden konnte. Auf dem Bedienfeld läßt sich diese Art der Minimierung durch den Schalter *look* im Block für die zu verwendende Minimierungsmethode einstellen, vgl. Abschnitt 6.5 .

- **Steilster Abstieg:** Hierbei muß zunächst der Gradient der zu minimierenden Funktion bestimmt werden. Dies geschieht näherungsweise durch einen

Differenzenquotienten, wie bereits in Abschnitt 4.2.2 erwähnt. Dazu werden hier zwei Verfahren angeboten, wobei das erste nur eine einseitige Differenz bestimmt. Das bedeutet, jeder Parameter wird um einen bestimmten Betrag geändert, der Funktionswert berechnet, von dem Ausgangswert subtrahiert und durch die Schrittweite dividiert. Der auf diese Weise erzeugte Gradientenvektor wird anschließend normiert, um die Fortschreitungsrichtung zu erhalten. Dieses Verfahren wird mit dem Schalter *grad+* eingestellt. Ein fast gleiches Verfahren wird über den Schalter *grad+−* ausgewählt. Dieses Verfahren unterscheidet sich von dem vorhergehenden nur dadurch, daß Funktionswerte in positive und negative Parameterrichtung ermittelt werden und aus deren Differenz der Gradient bestimmt wird.

Einen wichtigen Einfluß auf die Verfahren hat die Schrittweite zur Bestimmung der Funktionswerte. Wird diese zu klein gewählt, so gelangt man sehr leicht in ein lokales Minimum. Zu diesem Zweck besitzt das Modul zwei Parameter, mit denen sich diese Schrittweite angeben läßt. Der Wert *sh_alpha* bestimmt hierbei die Schrittweite für die Verschiebungsparameter in *mm*, während *phi_alpha* die Schrittweite für die Rotationsparameter in Grad angibt. Eine Trennung dieser beiden Parametergruppen ermöglicht neben der Kompensation der verschiedenen Einflüsse auf die Funktionswertberechnung eine Betonung der einen oder anderen Gruppe. Analog zu den Parametern für die Schrittweite existieren die zwei Parameter *sh_beta* und *phi_beta*, die die minimale Änderung in Fortschreitungsrichtung für die beiden Parametergruppen angeben.

Nachdem die Suchrichtung bestimmt ist, fährt der Algorithmus gemäß den minimalen Änderungen in der Suchrichtung mit einem Anfangssuchschritt fort. Wird an dem Endpunkt dieses Schrittes eine Verbesserung festgestellt, verdoppelt sich die Schrittweite des Anfangssuchschrittes. Dieses Verfahren wird so lange fortgesetzt, bis keine weitere Verbesserung erreicht wird. Da die letzte Schrittweitenverlängerung zu einer Verschlechterung geführt hat, wird mit Hilfe der im vorletzten Schritt ermittelten Parameter ein Startvektor für eine neue Iteration bestimmt. Läßt sich mit neuen Iterationen keine weitere Verbesserung erreichen oder ist diese Verbesserung kleiner als ein im Algorithmus vorgegebener Wert, bricht das Verfahren ab und das Modul stellt den zuletzt ermittelten Vektor an der rechten Ausgangsschnittstelle zur Verfügung. Die linke Ausgangsschnittstelle transportiert hierbei die Liste der gedrehten und verschobenen PET-Punkte gemäß dem Ausgangsvektor.

Zu Testzwecken kann das Modul mit dem Schalter *test* in einen Modus geschaltet werden, in dem der empfangene Startvektor am Ausgang des Moduls weitergegeben wird, wobei die PET-Punkte entsprechend transformiert ausgegeben werden. Bei diesem Modus findet keinerlei Anpassung statt, er ist dazu geeignet, eine Bewertung des Startvektors oder eine Anpassung von Hand vorzunehmen.

Der Aufruf des Moduls lautet:

```
int find_transf_compute( mparmrin, mrin, petin, defparin, parout,
transfpetout, phi_alpha, phi_beta, sh_alpha, sh_beta, mode)
        param_type *mparmrin;     /*MRT-Maschinenparameter*/
        AVSfield_int *mrin;      /*MRT-Daten*/
        AVSfield_float *petin;  /*PET-Daten*/
        transf_type *defparin; /*Parameterschaetzwerte*/
        transf_type **parout; /*optimierte Parameter*/
        AVSfield_float **transfpetout; /*transformierte PET-Punkte*/
        float *phi_alpha;   /*Winkel fuer Differenzbildung*/
        float *phi_beta;   /*Winkel fuer Minimumsuche*/
        float *sh_alpha;  /*Verschiebung fuer Diff.bildung*/
        float *sh_beta;  /*Verschiebung fuer Minimumsuche*/
        char *mode;     /*gewaehlte Methode*/
{ ... }
```

Nach der Übergabe der MRT-Maschinenparameter sowie MRT- und PET-Daten wird mittels des Benutzerdatentyps *transf_type* der Startvektor für die Suche übergeben. Als nächstes folgt die Definition der Ausgangsparameter, die mit dem Ausgangsvektor für die Anpassung beginnt, gefolgt von der Liste der transformierten PET-Punkte für den Ausgabevektor. Die Liste der interaktiven Modulparameter besteht im ersten Teil aus den oben bereits erwähnten Parametern für die Differenzbildung und Suchvektorbestimmung des Minimierungsverfahrens. Als letzter Aufrufparameter wird eine Zeichenkette mit der vorgewählten Anpassungsmethode übergeben.

Neben der Ausgabe an den Ausgangsschnittstellen gibt dieses Modul auch die ermittelten Bewertungsparameter auf dem Textbildschirm in Form der durchschnittlichen linearen Abweichung aus. Für diese Abweichung wird zunächst die Summe der Beträge der Abweichungen bestimmt, die anschließend auf die Anzahl der beteiligten Punkte bezogen wird. Als zweite Größe wird die Summe der Abweichungen zum Quadrat, bezogen auf die Anzahl der Punkte ausgegeben. Aus diesen beiden Größen wird anschließend die Standardabweichung ermittelt und ebenfalls ausgegeben. Neben diesen Bewertungsparametern werden zum Schluß auch die endgültigen Anpassungsparameter im Textfenster ausgegeben.

6.3.4 Datenneuanordnung

Nachdem die Drehungs- und Verschiebungsparameter von dem Modul *find_transf* bestimmt sind, ist es möglich, den PET-Datenschichten MRT-Informationen zuzuordnen. Abb. 6.12 zeigt hierzu eine mögliche Anordnung der Schichten im Profil. Die Neuanordnung übernimmt das Modul *reslice* aus Abb. 6.1.

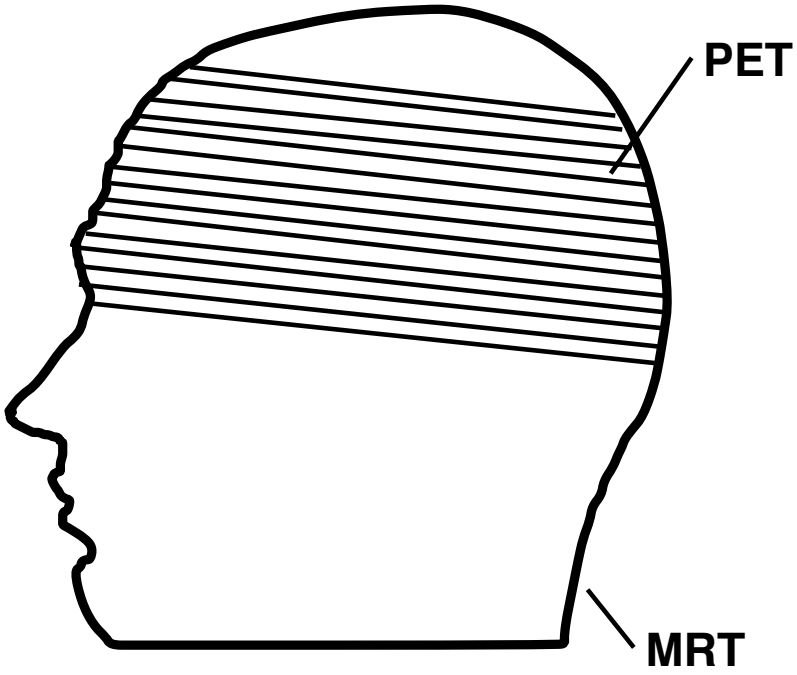


Abbildung 6.12: Lage der PET-Datenschichten im MRT-Datensatz

Um eine genaue Zuordnung von PET- und MRT-Daten zu erhalten, werden zwei Datensätze erzeugt, die die gleichen Dimensionen besitzen. Ein Beispiel für diese Datensätze findet sich in Abb. 6.13, wobei auf der linken Seite eine Schicht des PET-Datensatzes und auf der rechten Seite die zugehörige MRT-Datenschicht abgebildet ist.

Neben den drei Dateneingängen besitzt das Modul drei Eingabeparameter, die die relative Vergrößerung der Ausgabedatensätze in Relation zu den PET-Datensatzausmaßen angibt. Sind alle diese Werte 1, so haben die erzeugten Datensätze die gleiche Dimension wie der originale PET-Datensatz. Wird eine relative Vergrößerung angegeben, die größer als 1 ist, so werden entsprechend größere Datensätze generiert. Bei einem Faktor von 2 werden demnach doppelt so viele Punkte für die Ausgabedatensätze erzeugt, als ursprünglich für die PET-Daten vorhanden waren. Für den ausgegebenen PET-Datensatz bekommt man zwar nicht mehr Information, aber die Dimension der Daten entspricht der Dimensionierung des ebenfalls erzeugten MRT-Datensatzes, der dann mehr Information bietet, da dieser Ausgangsdatensatz höher dimensioniert ist als der PET-Datensatz. Es ist nicht sinnvoll, die Dimensionierung wesentlich größer als die maximale Auflösung der MRT-Daten zu wählen, da dann Punkte des MRT-Datensatzes mehrfach in den Datensatz übernommen werden und so keine weitere Information liefern. Die Größe des Datensatzes würde so nur unnötig in die Höhe getrieben.

Der Ausgabedatensatz PET wird erzeugt, indem die PET-Daten entsprechend den Vergrößerungsfaktoren neu abgetastet werden. Für die Ermittlung der passenden MRT-Schichten erzeugt der Algorithmus intern die Einheitsvektoren eines Koordi-

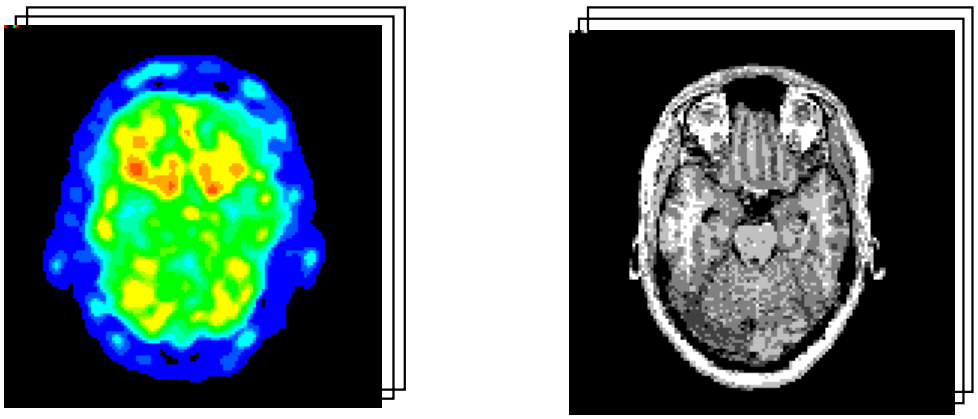


Abbildung 6.13: Neu erstellte PET- und MRT-Datensätze

natensystem der PET-Daten in der MRT-Welt, das bereits an die richtige Position verschoben und entsprechend rotiert worden ist. Sind diese Vektoren bestimmt, läßt sich anhand der Auflösung der PET- und MRT-Daten auf die entsprechenden Daten im MRT-Datensatz schließen. In diesem Zusammenhang sei erwähnt, daß bei der Ermittlung eines MRT-Datenwertes keine Interpolation stattfindet. Es wird das Datum übernommen, welches sich an einer berechneten Position befindet. Ist eine genauere Auflösung erwünscht, sind Datensätze größeren Ausmaßes zu erzeugen. Vor allen Dingen sollte dabei die Anzahl der erzeugten Schichten erhöht werden, da in dieser Raumrichtung für gewöhnlich die geringste Auflösung vorliegt. Sind auf diese Weise neue Schichten erzeugt worden, steht es dem Benutzer frei, aus dem hochauflösenden Datensatz niedrigerauflösende Schichten nach seinen Ansprüchen zu interpolieren.

Die beiden erzeugten Datensätze werden im Format *AVSfield_char* als Bytes weitergegeben. Als Eingangsdaten finden sich auf der linken Modulseite zunächst die Anpassungsparameter, gefolgt von den PET-Maschinenparametern und den eigentlichen PET-Daten. Die anschließenden Dateneingänge erhalten die MRT-Originaldaten und die diesen Datensatz beschreibenden Maschinenparameter. Die linke der beiden Modulschnittstellen gibt den erzeugten neuen PET-Datensatz weiter, während an der rechten Schnittstelle die zugehörigen MRT-Daten zur Verfügung stehen. Die Modulparameter sind in die Benutzeroberfläche integriert worden. Der Aufruf lautet:

```
int reslice_compute( mparmrin, mrin, petin, mparpetin, regpar, mrout,
petout, mag_x, mag_y, mag_z)
        param_type *mparmrin;              /* MR-Parameter */
        AVSfield_char *mrin;              /* MR-Eingangsdaten  */
        AVSfield_char *petin;            /* PET-Eingangsdaten */
        param_type *mparpetin;          /* PET-Parameter */
        transf_type *regpar;           /* Reslice-Parameter */
```

```
        AVSfield_char **mrout;          /* Ausgabe MR-Schichten */
        AVSfield_char **petout;        /* Ausgabe PET-Schichten */
        float *mag_x;                 /* Vergroesserung x-Richtung */
        float *mag_y;                /* Vergroesserung y-Richtung */
        float *mag_z;               /* Vergroesserung z-Richtung */
{   ...   }
```

6.4 Ausgabe

In diesem Abschnitt werden die implementierten Ausgabemechanismen und die zugehörigen AVS-Netzwerkteile näher untersucht. Die Einbettung der Ausgabe in die Benutzeroberfläche findet sich in Abschnitt 6.5 . Die Ausgabemodule finden sich in Abb. 6.1 unter den Namen *display_result* und *control_display*. Während *display_result* sich um die Ausgabe der neu erzeugten Schichten kümmert, bietet *control_display* eine Kontrollausgabe, die eine optische Bewertung des Anpassungsergebnisses zuläßt. Neben diesen beiden Ausgaben geben einige Module Meldungen im Textfenster aus, um damit z.B. eine Ausführung zu quittieren. Eine wichtige Textausgabe macht das Modul *find_transf*, vgl. Abschnitt 6.3.3, indem es die Anpassungsparameter sowie ein Bewertungsmaß für die Zuordnung ausgibt.

6.4.1 Bildausgabe

Das Makro-Modul *display_result* enthält die Module für die Darstellung der Schichtbilder auf dem Bildschirm. Hierbei wird nur eine Schicht des jeweiligen Datensatzes zu einer Zeit dargestellt. Neben den beiden Rohdatensätzen wird auch ein Überlagerungsbild angeboten.

Abb. 6.14 zeigt die Auflösung von *display_result*. Der linke Eingangsdatenpfad übernimmt das AVS-Feld mit den darin enthaltenen PET-Daten, während der rechte Datenpfad die MRT-Daten annimmt. Die Weiterverarbeitung dieser beiden Datenpfade sieht sehr ähnlich aus. Zunächst wird eine Schicht aus dem jeweiligen Datensatz mit dem *orthogonal slicer* geschnitten, wobei der Parameter für die Schichtnummer jeweils nach außen gelegt wurde und zentral über das Modul *integer* angesteuert wird. Somit werden zueinander passende Schichten erzeugt und weiterverarbeitet. Der Drehzeiger für das Modul *integer* ist in die Bedienungsoberfläche integriert. Die erzeugten Schichten werden jeweils mit dem Modul *colorizer* eingefärbt. Für die PET-Daten wird hierbei eine Farbtafel verwendet, während die MRT-Daten nur schwarz/weiß eingefärbt werden. Am Ausgang dieser Module stehen die eingefärbten Bilder als Datensätze zur Verfügung, die direkt mit den Modulen *display image* angezeigt werden können.

Die Kombination der beiden Bilder übernimmt das für diesen Zweck erstellte Modul *combine_pet_mr* . Zu diesem Modul existiert ein Modulparameter, der den Überblendfaktor zwischen den eingehenden Datensätzen regelt. Dieser Regler ist auf

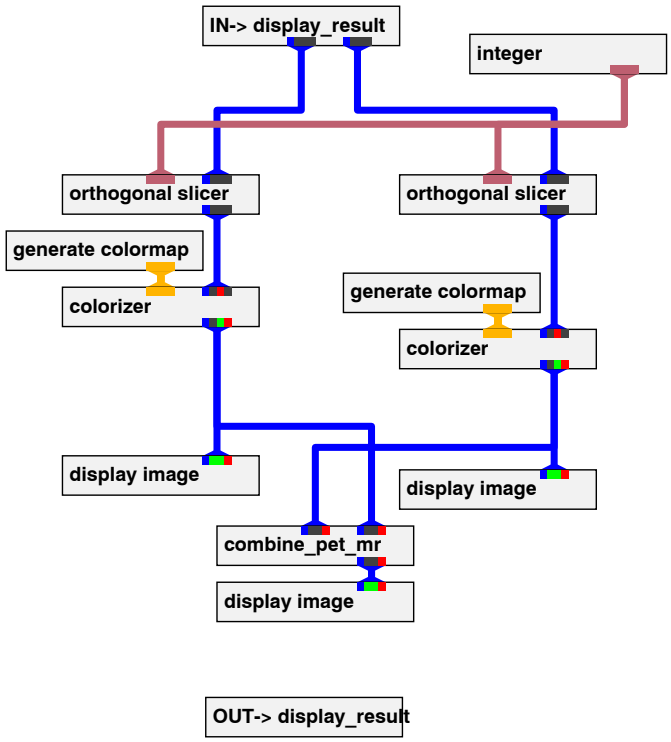


Abbildung 6.14: Netzwerk für die Bildausgabe

die Bedienoberfläche herausgeführt. Wird dort der Wert 0 angegeben, wird nur das MRT-Bild dargestellt, bei jedem anderen Wert wird das Bild der PET-Daten anteilmäßig beigemischt. Dieses so erzeugt Bild wird mit einem weiteren Modul *display image* dargestellt. Beispiele für die Darstellung finden sich in Abb. 6.16 .

6.4.2 Kontrollausgabe

Neben den bereits angesprochenen Ausgaben im Textfenster für die Bewertung des Anpassungsergebnisses generiert das Makro-Modul *control_display* einen Kontrolldatensatz, der von dem AVS-Modul *geometry viewer* dargestellt wird.

Abb. 6.15 zeigt das in *control_display* verborgene Netzwerk. Die linke Eingangsschnittstelle übernimmt die MRT-Maschinenparameter, während die mittlere Schnittstelle die MRT-Konturen aus dem Modul *gen_cont* übernimmt. Diese beiden Datenströme werden dem Modul *pet_list_to_points* übergeben, der genau wie bei den PET-Konturen nun eine Reihe von MRT-Punkten im Raum generiert, vgl. Abschnitt 6.3.2 . Wie bereits dort erwähnt, findet eine Transformation der Punkte von einem PET- in ein MRT-Koordinatensystem statt. Da die MRT-Konturen aber bereits auf das MRT-Koordinatensystem bezogen waren, muß die Transformation von PET nach MRT rückgängig gemacht werden. Diese Aufgabe übernimmt das Modul *mr_to_pet_world*. An diesem Ende der Verarbeitung erhält man eine Liste von Punkten im MRT-Raum, die die Außenkontur der MRT-Daten charakterisieren. Diese

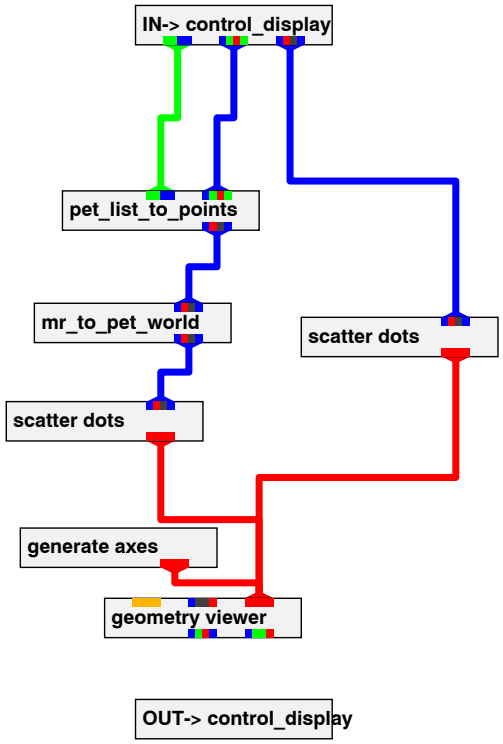


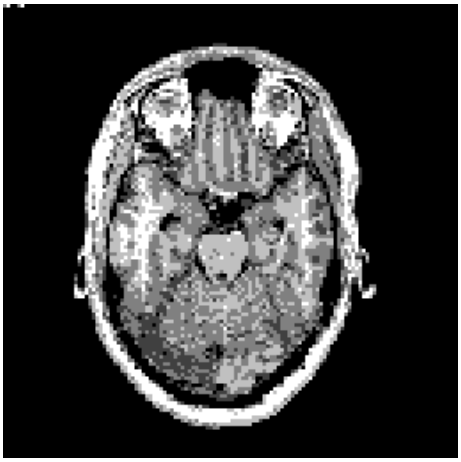
Abbildung 6.15: Netzwerk für die Kontrollausgabe

Punkte werden dem AVS-Modul *scatter dots* zugeführt, der daraus Geometriedaten für den *geometry viewer* erzeugt, die dann als kleine Kugeln um jeden Punkt dargestellt werden.

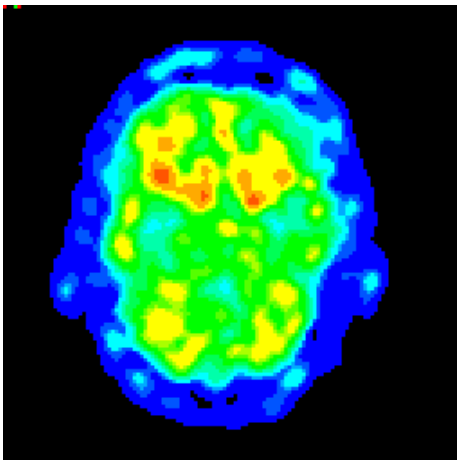
Der rechte Makro-Moduleingang erhält die bereits vorher erzeugte Liste angepaßter Punkte, die die Außenkontur der PET-Daten repräsentieren, wie sie nach der Anpassung im Raum liegt. Diese Punkte werden von dem angeschlossenen Modul *scatter dots* untereinander verbunden und ebenfalls als Geometriedaten an den *geometry viewer* übergeben. Zur besseren Orientierung stellt der *geometry viewer* auch ein von *generate axes* übernommenes Koordinatenkreuz dar. Abb. 6.16 zeigt rechts unten ein Bild dieser Kontrollausgabe, wobei die Lage der PET-Daten (durchgezogene Linien) relativ zu den MRT-Daten (Punkte) erkennbar ist. Für die Benutzung des *geometry viewer* sei hier auf [3] verwiesen.

6.5 Bedienungselemente

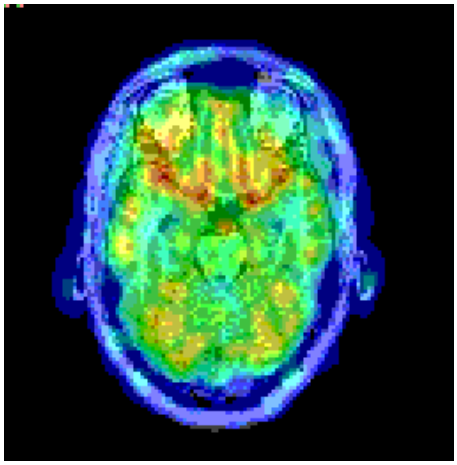
In diesem Absatz werden die Bedienungselemente des gesamten Netzwerkes in den dafür erstellten Fenstern gezeigt. Diese Fenster stellen die Oberfläche und den Zugang für den Bediener dar. Die zur Verfügung gestellten drei Bedienfenster sind in den Abb. 6.17 bis 6.19 abgebildet und werden hier der Reihe nach vorgestellt. Die eingestellten Werte für die Bedienungselemente gelten für die Anpassung des in



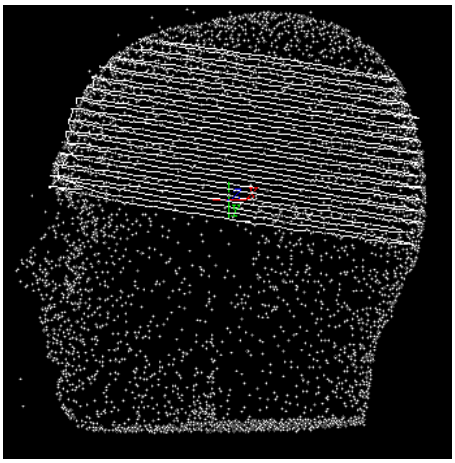
MRT



PET



Überlagerung



Kontrollbild

Abbildung 6.16: Ausgabebilder

dieser Arbeit verwendeten Beispieldatensatz.

In dem oberen Teil des Hauptfensters mit dem Namen *AVS main* finden sich die Ausgabeeinheiten der Bildausgabe in Abschnitt 6.4.1 . Im linken Anzeigeelement befindet sich die angewählte PET-Datenschicht, gefolgt von der passenden MRT-Datenschicht. Das letzte Bild zeigt die Überlagerung dieser beiden Bilder. Im mitteleren Teil des Fensters befinden sich zwei Drehzeiger, die ebenfalls zur Bildausgabe gehören. Mit dem Drehzeiger *slice* läßt sich die Nummer der Schicht wählen, die angezeigt werden soll. Mit dem Überblendfaktor *blend* läßt sich die Überblendung von PET- und MRT-Schicht regeln. Wird der Wert 100 eingestellt, so ist nur das PET-Bild zu sehen. In der linken unteren Ecke des Fensters befinden sich die Bedienungselemente für das Startwertmodul, wobei sich hier der Startvektor für die

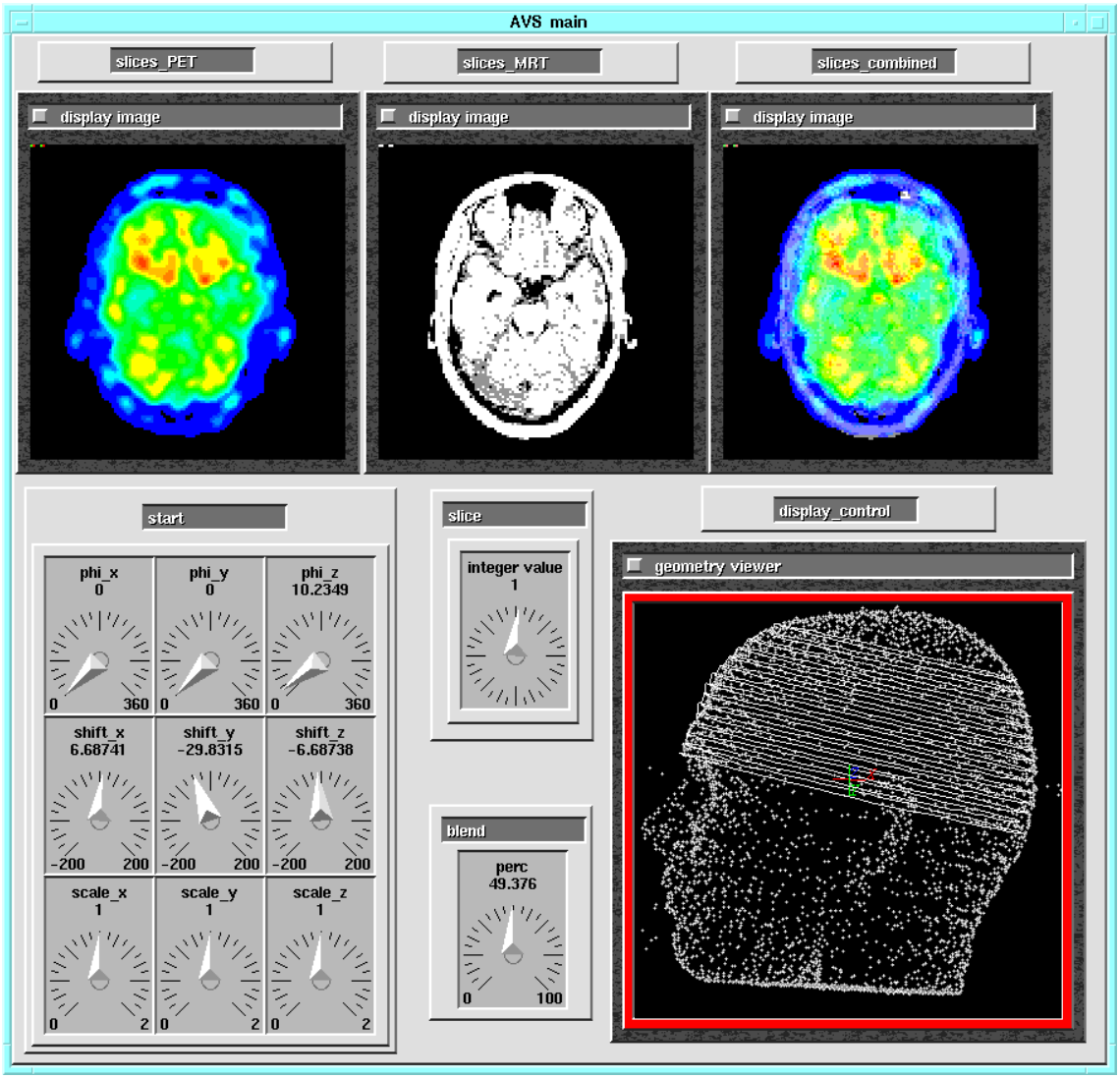


Abbildung 6.17: Bedien- und Ausgabefenster *AVS main*

Anpassung aus Abschnitt 6.3.3 wählen läßt. Innerhalb dieses Bedienungsfeldes existieren Drehzeiger für die Werte der Drehwinkel um x-, y- und z-Achse, gefolgt von den Werten für die Verschiebung in den entsprechenden Richtungen. Die Faktoren für die Skalierung sind zwar hier aufgeführt, werden vorerst aber nicht weiter genutzt. In der rechten unteren Ecke befindet sich die Ausgabe des *geometry viewer* aus dem Kontrollausgabemodul in Abschnitt 6.4.2 . Die die PET-Schichten charakterisierenden Punkte sind hier untereinander verbunden, während die MRT-Punkte als kleine Kugeln dargestellt sind.

In dem Fenster *AVS control* in Abb. 6.18 finden sich die Elemente, die den gesamten Algorithmus in grundlegender Weise beeinflussen. In den Feldern *PET-file* und *MRT-file* in der linken oberen Ecke befinden sich die Auswahlelemente für die zu bearbeitenden Datensätze, wobei diese Elemente zu den Dateneinlesemodulen *read_PET_data* und *read_MR_data* aus Abschnitt 6.2 gehören.

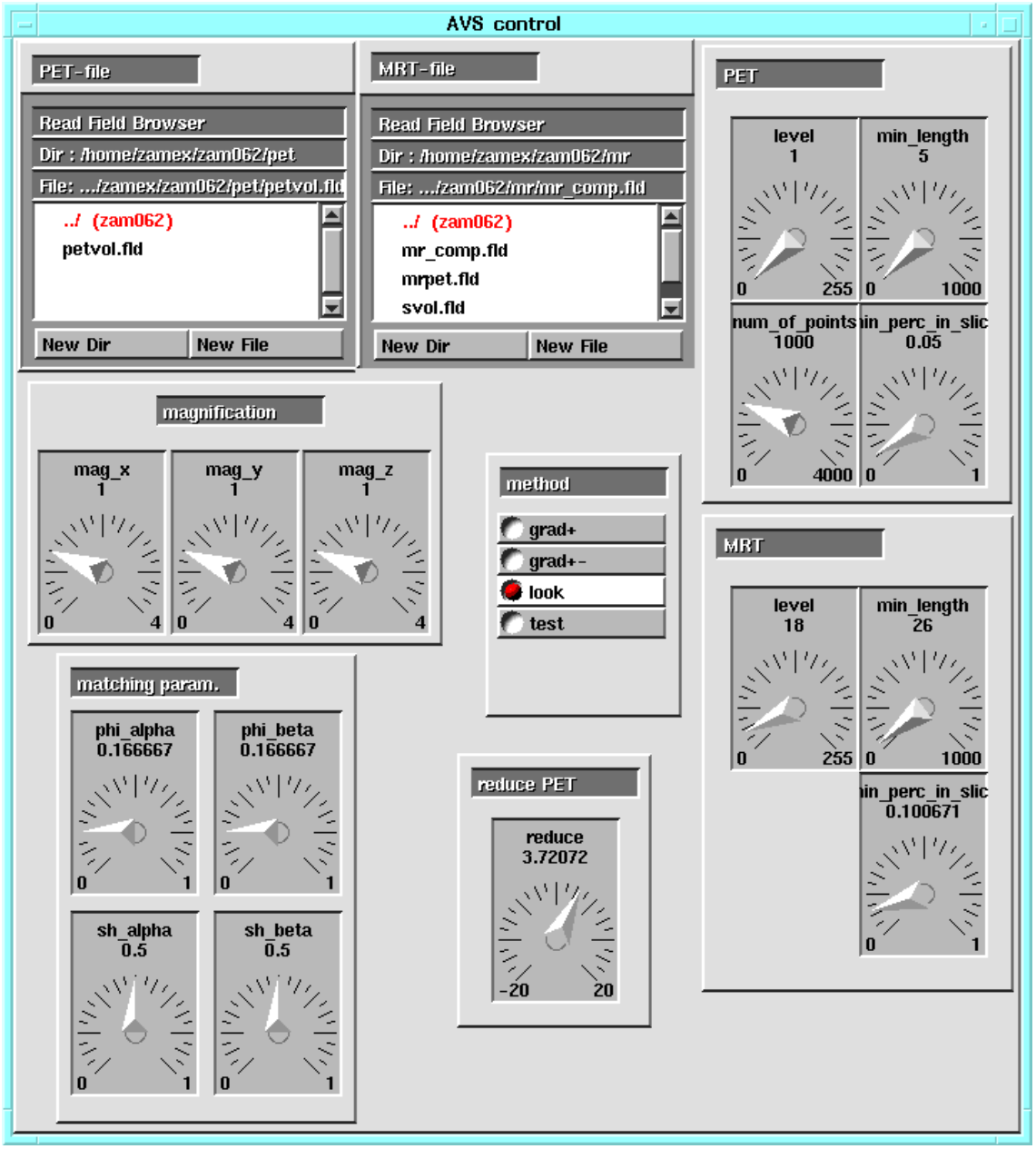


Abbildung 6.18: Bedien- und Steuerfenster *AVS control*

Das Feld *PET* in der rechten oberen Ecke beherbergt die für die PET-Vorverarbeitung wichtigen Parameter. Der Drehzeiger *level* gehört zum Schwellwertmodul der PET-Vorverarbeitung, wie es in Abschnitt 6.3.1 für die MRT-Vorverarbeitung beschrieben ist. Die Parameter *min_length*, *min_perc_in_slice* und *num_of_points* gehören zum Modul *pet_list_to_points* aus Abschnitt 6.3.2 und beeinflussen die PET-Punkteerzeugung. Die Anzahl der gewünschten PET-Punkte kann hier mit dem Drehzeiger *num_of_points* eingestellt werden. Wie bereits bei der PET-Punkterzeugung erwähnt, gibt es zwei Kriterien, eine Kontur innerhalb einer erstellten Konturliste auszuwählen. Das erste Kriterium, daß eine Kontur eine Mindestlänge von Punkten hat, wird durch *min_length* beeinflußt, die rela-

tiv zur längsten Kontur benötigte Länge durch *min_perc_in_slice*. Ein weiteres der PET-Vorverarbeitung zuzuordnendes Feld findet sich im unteren Bereich des Fensters mit dem Namen *reduce PET*. Dieser Regler gehört zum Modul *pet_reduce* aus Abschnitt 6.3.2 und gibt die Größe der Verkleinerung der PET-Kontur in *mm* an. Positive Werte bedeuten eine Verkleinerung, negative eine Vergrößerung der Kontur.

Unter dem Feld *PET* findet sich das Feld für vergleichbare Parameter der MRT-Vorverarbeitung namens *MRT*. Auch hier kann der Wert für die Schwellwertoperation interaktiv eingegeben werden. Da das Modul *mr_list_to_surface* aus Abschnitt 6.3.1 als Parameter nur die Auswahlparameter für die minimale Länge einer Kontur besitzt, fehlt ein Parameter für die Anzahl der Punkte.

Für die Anpassung aus Abschnitt 6.3.3 gibt es zwei Parameterfelder. Mit dem Feld *method* läßt sich die Optimierungsmethode vorgeben, nach der eine Anpassung bestimmt werden soll. Das zweite Feld trägt den Namen *matching param.* und gibt die Werte für das Anpassungsmodul an. Die Angaben sind hier in Grad bei den *phi*-Werten und *mm* bei den *sh*-Werten. *alpha* kennzeichnet die Werte für die Bestimmung des Differenzenquotienten und *beta* die Werte für die Fortschreitung in Suchrichtung.

Das Feld *magnification* gehört zum Modul *reslice* aus dem Abschnitt 6.3.4 über die Datenneuanordnung. Die Faktoren *mag_x* bis *mag_z* geben die Vergrößerung des Datensatzes in die entsprechende Raumrichtung an. Hierbei ist das Bezugskoordinatensystem im Gegensatz zum sonst üblichen Bezugssystem MRT die PET-Welt.

In Abb. 6.19 finden sich die Bedienungselemente, um die Maschinenparameter einzustellen, die nicht im Datensatz enthalten sind. Dieses Fenster gehört zu den Modulen *PET_parameter* und *MRT_parameter* aus Abb. 6.1. Im oberen Bereich werden die Größen eines Datenpixels für jede Raumrichtung in *mm* angegeben. Die Zuordnungen der Raumrichtungen gelten hier für die jeweiligen Datensätze wie in Abschnitt 6.2 beschrieben. Die nächste Zeile gibt den minimalen Wert an, der für die jeweilige Raumrichtung als Index berücksichtigt werden soll. Die letzte Zeile enthält die Angaben für den maximal zu berücksichtigenden Pixelindex in der jeweiligen Raumrichtung.

Es existiert noch eine Reihe weiterer Parameter zu anderen Modulen, die allerdings hier nicht weiter erwähnt werden, da sie nicht Bestandteil der Benutzeroberfläche sein sollen und auf feste Werte eingestellt wurden. Sollten diese Parameter aus irgendeinem Grunde trotzdem interessant werden, so sind sie über den Netzwerk-Editor des AVS einfach zu erreichen. Die hier vorgestellten Parameter reichen jedoch für die Handhabung des Verfahrens vollkommen aus.

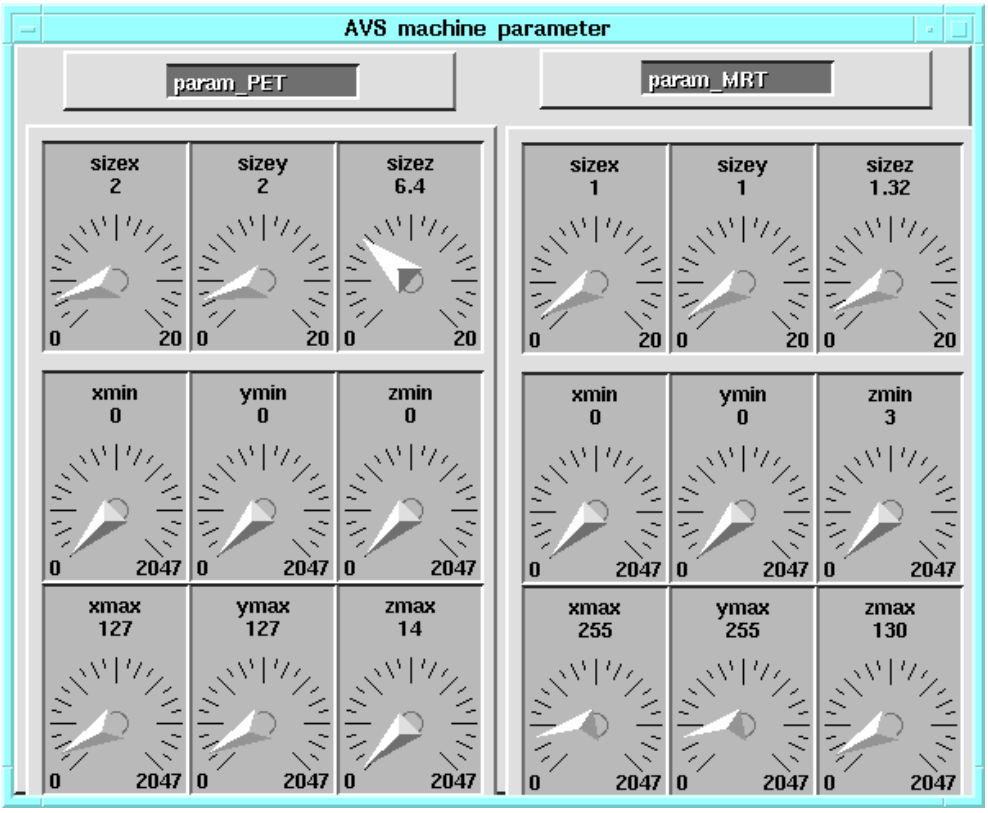


Abbildung 6.19: Eingabefenster für Maschinenparameter *AVS machine parameter*

6.6 Ergebnisse des sequentiellen Ansatzes

Neben den Ergebnissen bzgl. des verwendeten Datensatzes werden in diesem Abschnitt erste Erfahrungen mit dem Gesamtsystem gegeben.

Die hier verwendeten iterativ rekonstruierten PET-Daten haben sich als brauchbar für eine Konturgenerierung erwiesen. Hierbei ist der Schwellwert für die Klassifizierung der Daten in Objekt oder Hintergrund sehr niedrig anzusetzen, ansonsten erhält man zwar kleinere Konturen, die jedoch mit Einbuchtungen aufweisen, die häufig nur noch innere Strukturen wiedergeben. Die mit niedrigem Schwellwert erhaltenen Konturen liegen aufgrund der Auflösung des PET-Verfahrens weiter außen als die eigentliche Oberfläche des Kopfes. Dieser Effekt wird durch die Auflösungskompensation aufgefangen, die in dieser Arbeit entwickelt wurde. Bei höherer Auflösung kann diese Kompensation zunehmend wegfallen.

Hat man funktionale Datensätze, bei denen sich nur innere Strukturen definieren lassen, so können für das hier gezeigte Verfahren weitere Module implementiert werden, die es ermöglichen, Punkte der Außenkontur aus anderen Modellen heraus zu entwickeln. Sind diese Punkte vorhanden, steht der Ausführung des weiteren Verfahrens nichts im Wege, da die erzeugten Punkte direkt mit bestehenden Modulen weiterverarbeitet werden können.

Bei der Schwellwertbildung der MRT-Daten stößt man auf das Problem, daß mit kleiner werdendem Schwellwert die Übernahme von Störungen und Artefakten ins Binärbild größer wird. Diese Störungen lassen sich aber zum großen Teil dadurch kompensieren, daß man für die hier erstellte Tiefenkartenerstellung nur hinreichend lange Konturen berücksichtigt. Im allgemeinen ist bei der Schwellwertbildung für MRT auch ein geringer Schwellwert anzusetzen, um nicht Teile der Außenkontur schon vorab wegzuschneiden.

Aufgrund der beliebigen Festlegung der Abstände innerhalb der Datensätze durch die Angabe der Maschinenparameter ist es möglich, in der Größe variable Datensätze aneinander anzupassen, wobei nur jeweils die Maschinenparameter geändert werden müssen. Aus diesem Grunde wird eine mögliche Skalierung der Daten in den Raumrichtungen nicht weiter beachtet.

Nach der Vorverarbeitung findet die eigentliche Anpassung statt. Hierbei zeigt sich, daß die Ermittlung bzw. der Ausgleich einer Verschiebung vergleichsweise einfach erkannt wird, aber vorhandene Rotationen schwer entdeckt werden, was zu einem nicht unerheblichen Teil an Mehrdeutigkeiten bei der Anpassung der Daten liegt. Diese Mehrdeutigkeit erklärt sich durch am Schädel vorhandene Symmetrien, die auch anderen Verfahren Probleme bereiten [8].

Aufgrund der Diskretisierung der MRT-Daten, sowie der diskreten PET-Punkte für die Anpassung ergeben sich lokal sehr viele Minima, die übersprungen werden müssen. Dies ist neben der Verschiedenartigkeit der Parameter der Grund für die hier vorgeschlagene Einführung einer Mindestschrittweite für die Fortschreitungsrichtung und die Differenzberechnung. Desweiteren ist es möglich, mit diesen Parametern eine Betonung einer Parameterart durchzuführen. Aufgrund der weiteren Entwicklung dieser Arbeit blieb keine Zeit, diese Effekte weiter und vor allen Dingen an anderen Datensätzen zu untersuchen, wobei sich besonders für die Entwicklung geeigneter Optimierungsstrategien eine Reihe von Möglichkeiten bietet.

Bisher wurden nur allgemeine Optimierungsstrategien verfolgt, die nahezu keine oder nur sehr wenig Annahmen über die vorhandenen Daten machen. Es wäre durchaus denkbar, eine Optimierung einzuführen, die abwechselnd nur die Verschiebungen oder Rotationen optimiert.

Derzeit werden Erfahrungen mit dem hier vorgestellten System im Institut für Medizin gesammelt, die sicherlich zu einer weiteren Verbesserung des Systems führen. Erste Anwendererfahrungen sind bereits bei der Entwicklung des Systems eingeflossen. Für den hier betrachteten Datensatz ist eine Anpassung bis auf 1,85 mm durchschnittliche lineare Abweichung der PET-Punkte von der MRT-Oberfläche erreicht worden, die vergleichbar mit vorherigen Arbeiten mit diesem Datensatz sind [8]. Am Institut für Medizin sind mit dem im Rahmen dieser Arbeit implementierten Verfahren und anderen Datensätzen bereits Anpassungen bis 1,5 mm Abweichung gelungen. Abb. 6.20 zeigt die bereits vorhandenen 15 PET-Schichten, denen die neu errechneten 15 MRT-Schichten aus Abb. 6.21 zugeordnet wurden. In Abb. 6.22 ist die Überlagerung dieser Daten dargestellt.

Die interaktiven Möglichkeiten des AVS erlauben es dem Benutzer, dem Netzwerk an jeder Schnittstelle Daten zu entziehen, um so den Bearbeitungsprozeß transparent zu machen und wenn nötig einzugreifen. Dies schließt auch die Möglichkeit ein, die erhaltenen Daten in nachfolgenden Netzstrukturen weiterzuverarbeiten.

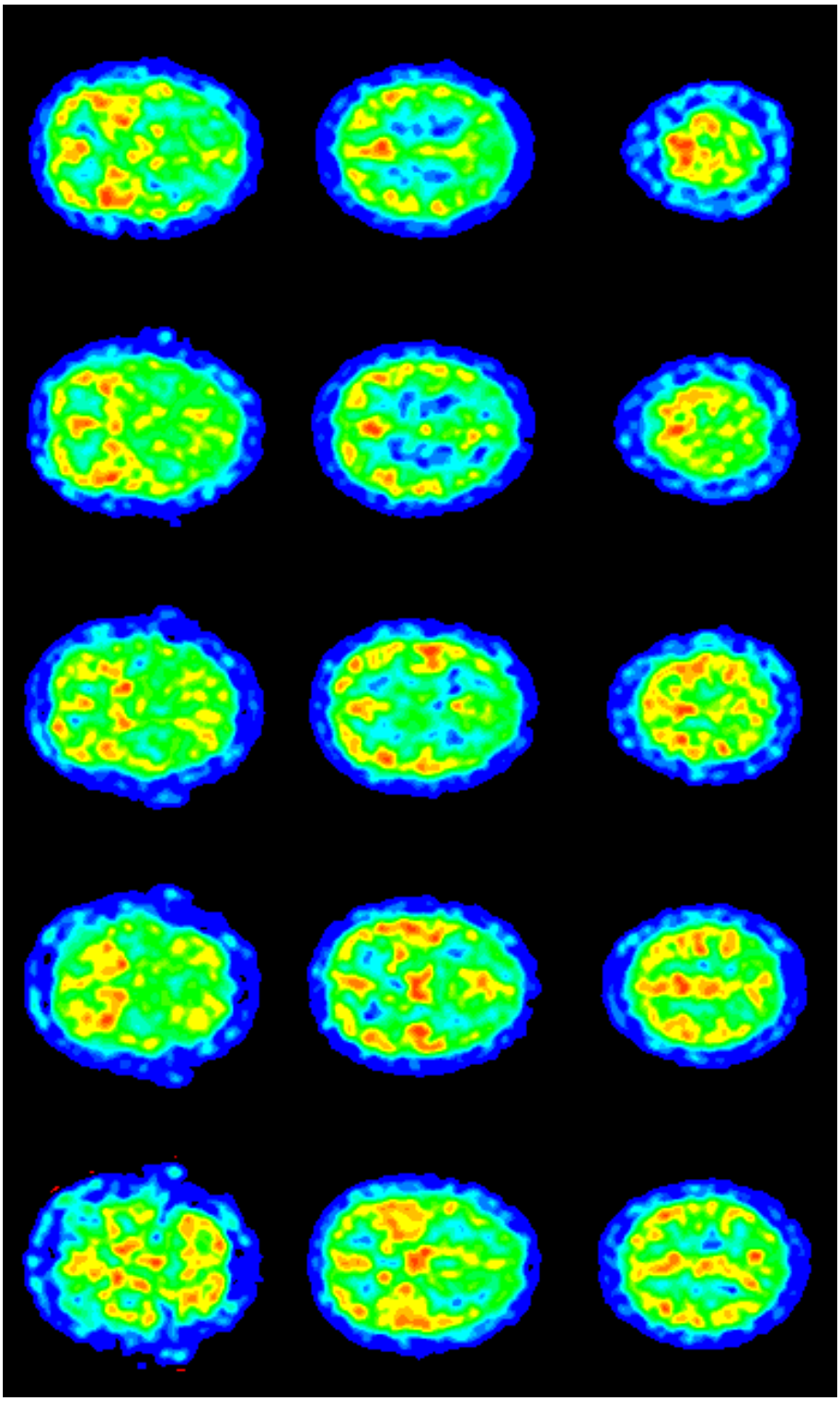


Abbildung 6.20: Übersicht des PET-Ausgangsdatensatz

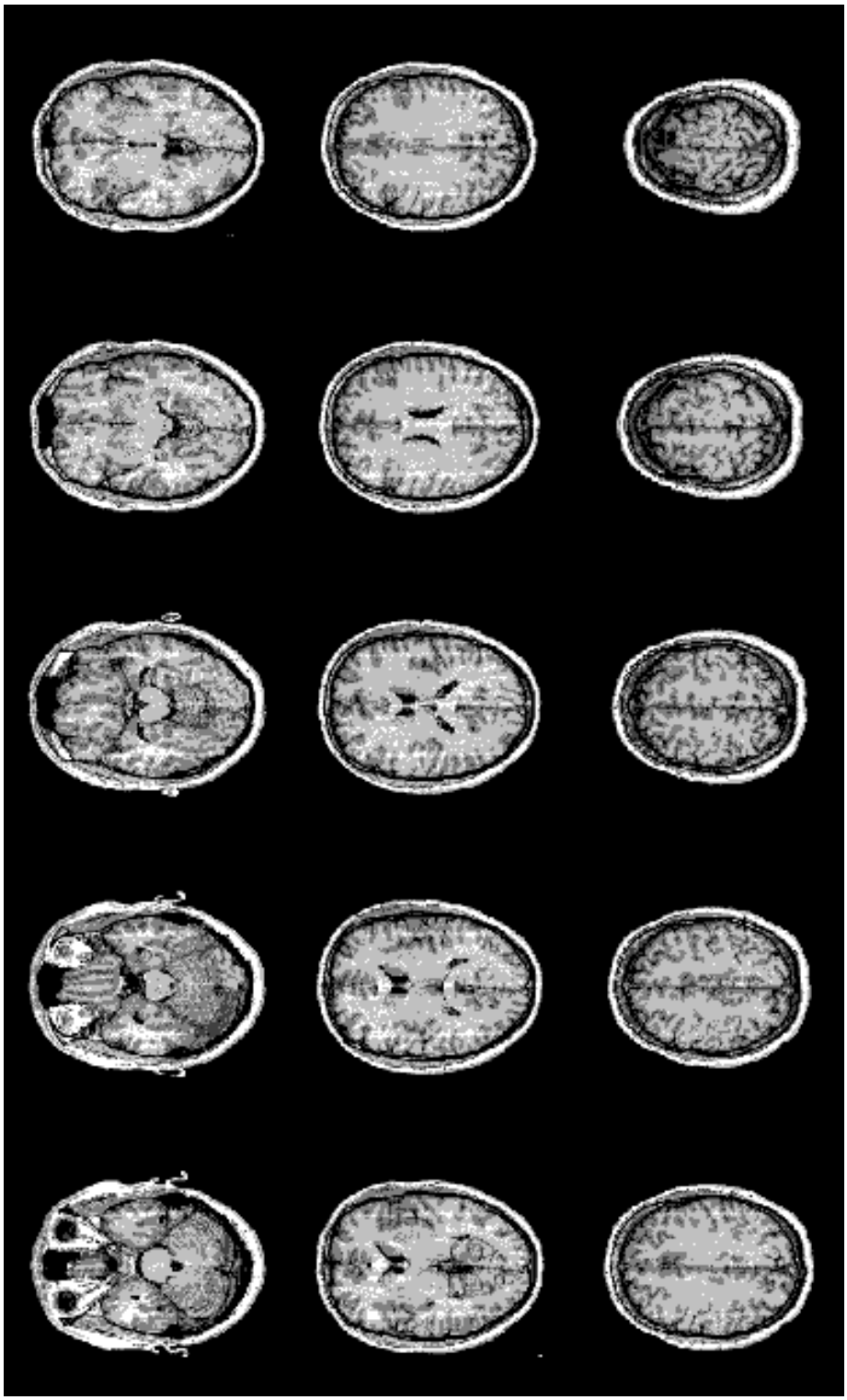


Abbildung 6.21: Übersicht berechneter und zugeordneter MRT-Schichten

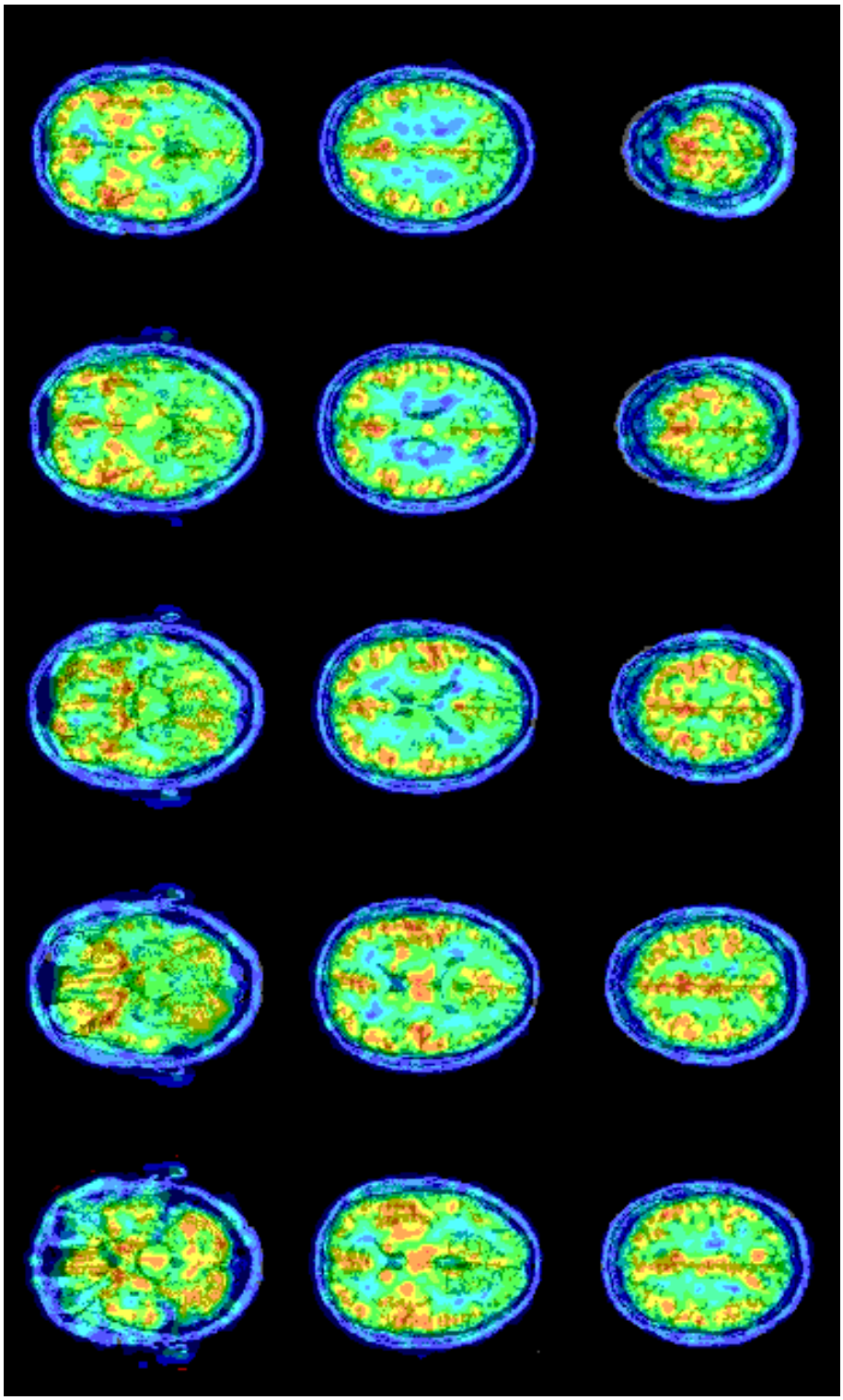


Abbildung 6.22: Überlagerung berechneter MRT-Schichten und PET-Schichten

Kapitel 7

Client/Server-Ansatz

Neben der im vorangegangenen Kapitel beschriebenen sequentiellen Realisierung des Anpassungsalgorithmus ist im folgenden daran gedacht, dieses Verfahren auf einem Parallelrechner zu implementieren. Normalerweise muß sich der Anwender des Programms bei einem Übergang auf ein paralleles System zuerst an die neue Bedienungsumgebung gewöhnen. Zusätzlich tritt häufig das Problem auf, daß ihm dort bereits bekannte Werkzeuge nicht zur Verfügung stehen. Aufgrund dieser Problematik wird im folgenden ein spezielles Client/Server-Modell entwickelt. Es soll dem Benutzer den bereits bekannten Algorithmus mit der vorgestellten Bedienungsschnittstelle bieten, aber gleichzeitig die Vorteile einer andersartigen Rechnerarchitektur nutzen.

7.1 Client/Server-Modell

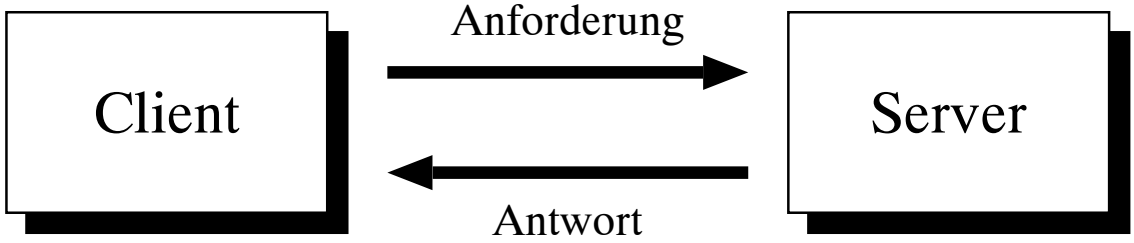


Abbildung 7.1: Client/Server-Modell

Bei dem bereits angesprochenen Client/Server-Modell [38][45] handelt es sich um ein Standardmodell der Netzwerkanwendung. Es beschreibt die Art, wie zwei Prozesse auf einem oder mehreren Rechnern miteinander kommunizieren. Abb. 7.1 zeigt dieses Modell im Überblick. Der Server stellt einen Prozeß in einen beliebigen Rechner dar, der auf Dienstanforderungen eines Client wartet. Hierzu wird

der Server-Prozeß in einem Rechner gestartet. Mit der stattfindenden Initialisierung des Server hat dieser einen Kommunikationskanal aufgebaut, an dem er unter einer vorgegebenen Adresse Aufträge eines Client entgegennehmen kann. Solange keine Anforderung eines Client auftritt, befindet sich der Prozeß im Wartezustand. Schickt ein Client über den aufgebauten Kommunikationskanal eine Dienstanforderung, so wird der Server aktiv, nimmt die Anforderung entgegen, bearbeitet sie in geeigneter Weise und sendet eine Antwort an den Client. Nach Beendigung der Client/Server-Verbindung geht der Server wieder in den Wartezustand und ist für eine neue Dienstanforderung bereit. Der Client stellt einen Prozeß dar, der Aufträge erstellt und an den Server schickt. Dieser Prozeß kann in derselben Maschine wie der Server-Prozeß laufen, jedoch ist es auch möglich, diese Prozesse auf verschiedenen Rechnern zu starten, wobei dann eine Kommunikation über ein verbindendes Netz stattfindet. Im allgemeinen ist es möglich, nicht nur einen, sondern eine Reihe von Client- und Server-Prozessen zu haben, wobei jeder Client an jeden Server einen Auftrag schicken kann. Wichtig ist dabei, daß die beiden Partner sich untereinander verstehen, also die gleiche Schnittstelle besitzen.

Die in dieser Arbeit vorgenommene Implementierung des Client/Server-Konzepts benutzt einen Teil bereits vorhandener Routinen aus [38], die es ermöglichen, eine Client/Server-Anwendung zu realisieren, ohne detaillierte Kenntnisse über die Programmierung sehr tiefer und komplexer Netzwerkroutinen zu besitzen. Die zur Verfügung gestellten C-Routinen teilen sich in Algorithmen, die für die Implementierung des Modells gedacht sind, und Routinen, die den Datenaustausch zwischen den beiden Prozessen erleichtern.

7.1.1 Verbindungsaufbau

Der zeitliche Verlauf einer Client/Server-Verbindung mit Hilfe der im folgenden beschriebenen Funktionen ist in Abb. 7.2 gezeigt. Bevor es möglich ist, einen Client-Auftrag zu bearbeiten, muß zunächst ein Server gestartet werden, der diesen Auftrag bearbeiten wird. Neben der Initialisierung als Prozeß in einer Maschine, muß der Server eine Schnittstelle aufbauen, über die der Server von außen zugänglich ist. Die hierfür vorgesehene Routine nennt sich

int **CreateService** *(port)*
int port;

Mit dem Aufrufparameter weist man dem Server eine Port-Nummer zu, unter der der Server zu erreichen ist. Neben der Port-Nummer muß lediglich die Adresse des Rechners im Internet[1] bekannt sein, um eine Verbindung herzustellen. Die Port-Nummer kann beliebig gewählt werden, sollte aber größer als 5000 sein, da kleinere Nummern bereits für die Systemverwaltung reserviert sind. Die Funktion gibt einen Deskriptor *sd* zurück, der der Server-Schnittstelle zugeordnet ist.

[1]Rechnernetzwerk für UNIX-Systeme, siehe [46]

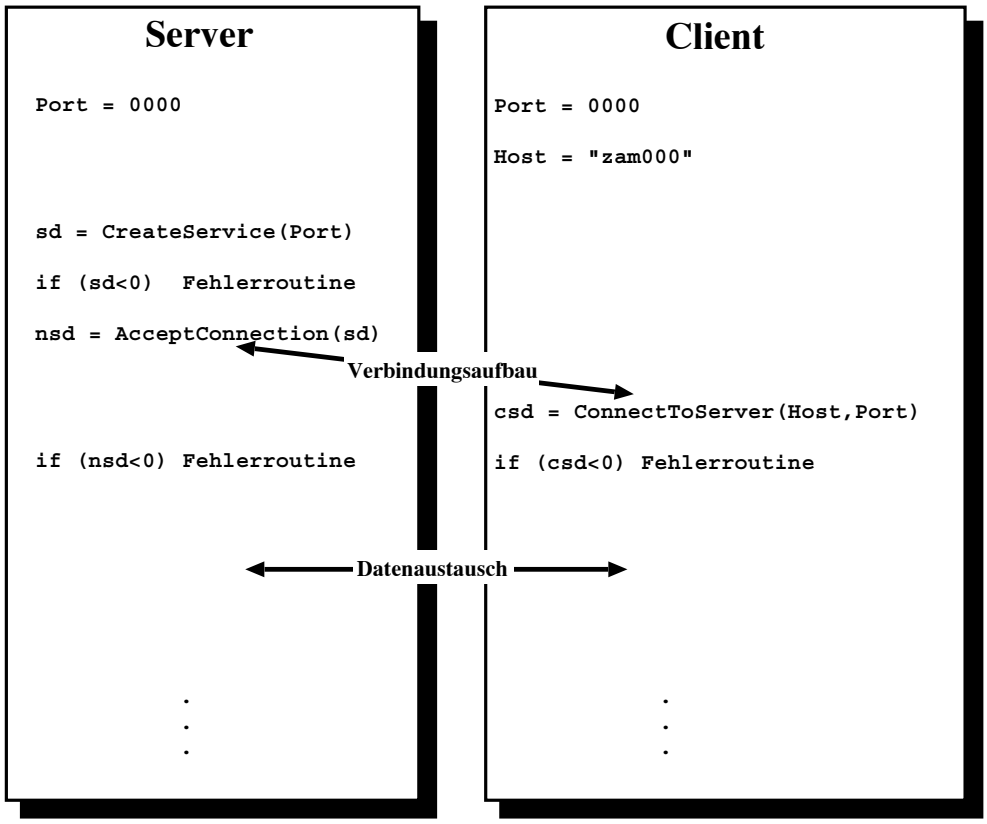


Abbildung 7.2: Zeitliche Ablauf einer Client/Server-Verbindung

Ist die Server-Schnittstelle erfolgreich definiert, kann der Server auf Anfragen eines Client warten. Die hierfür zur Verfügung stehende Funktion heißt

int **AcceptConnection** *(sd)*
int sd;

Der Aufrufparameter dieser Routine ist der vorher definierte Deskriptor *sd*. Diese Routine wartet, bis ein Verbindungswunsch eines Client eintrifft. Der Rückgabewert ist ein neuer Deskriptor *nsd*, über den die Kommunikation des Server mit dem Client in beide Richtungen möglich ist. Über den ursprünglichen Deskriptor *sd* können gleichzeitig weitere Aufträge angenommen werden.

Mit den beiden vorangegangenen Funktionen wird der Server für Aufträge eines Client bereit gemacht. Im folgenden wird die Dienstanforderung des Client betrachtet. Zu diesem Zweck ist die Routine

int **ConnectToServer** *(ServerName, port)*
*char *ServerName;*
int port;

definiert. Sie ermöglicht dem Client, eine Verbindung mit dem Server aufzunehmen. Als Aufrufparameter wird der Name des Rechners, auf dem sich der Server befindet, als Zeichenkette übergeben. Der zweite Aufrufparameter enthält die im Server gewählte Port-Nummer, die dem Client bekannt sein muß. Der Rückgabewert ist ein Deskriptor *csd*, über den das Client-Programm Daten mit dem Server austauschen kann. Diese Routine stellt somit den Beginn der Dienstanforderung des Client dar und veranlaßt den Server, den Auftrag anzunehmen und zu bearbeiten.

Die betrachteten Routinen geben einen Ausgabewert kleiner als 0 zurück, wenn ein Fehlerfall aufgetreten ist. Zum Schluß der Begegnung werden die erzeugten Deskriptoren *nsd* und *csd* wieder zerstört, um sie für einen weiteren Gebrauch freizugeben. Der Deskriptor *sd* wird gelöscht, wenn der Server beendet wird oder keine Anforderungen mehr annimmt.

7.1.2 Nachrichtenaustausch

Neben dem Aufbau einer Verbindung, werden Daten vom Client zum Server bzw. in die entgegengesetzte Richtung transportiert. In dieser Arbeit wurden dazu neben eigenen Datentransferroutinen auch bereits bestehende Routinen benutzt, die einen Datenaustausch in Form von Nachrichten ermöglichen. Da die beiden beteiligten Prozesse wechselseitig sowohl Sender als auch Empfänger sein können, sind diese Routinen für Client und Server nutzbar.

Ein Problem des Datentransfers zwischen Systemen unterschiedlicher Architektur ist eine unterschiedliche Datendarstellung auf den Rechnern. Eine Lösung für derartige Probleme bieten externe Datenrepräsentationen. Für diese Repräsentation wird die Beschreibungssprache XDR [38][44]verwendet. Ein Datentransfer mittels XDR teilt sich in mehrere Phasen. Zunächst werden die Daten des sendenden Rechners in einen sogenannten XDR-Stream umgesetzt, der die Daten in einem rechnerunabhängigen Format enthält. Dieser Vorgang wird als Serialisieren bezeichnet. Nach der Übertragung müssen die Daten in das Format des Zielrechners umgewandelt werden. Dies wird als Deserialisierung bezeichnet. Für die Serialisierung und Deserialisierung ist zwar eine Reihe von Standardfiltern für Umwandlung von Standarddatenstrukturen bereits vorhanden, jedoch müssen eigene Filter für benutzereigene Datentypen geschrieben werden. Hierbei werden die Filter so implementiert, daß sie sowohl für die Serialisierung als auch für die Deserialisierung geeignet sind.

Zur einfacheren Handhabung der rechnerunabhängigen Datenübertragung werden in [38] spezielle Sende- und Empfangsanweisungen bereitgestellt, die einen Austausch von Daten beliebiger Struktur ermöglichen. Die Routine zum Senden einer Datenstruktur lautet

int **SendXDR** *(dsd, ptr, xdr_dummy)*
int dsd;
char ptr;*
*int (*xdr_dummy());*

Der erste Aufrufparameter kennzeichnet den Deskriptor, über den eine Nachricht versandt werden soll. *ptr* stellt einen Zeiger auf eine zu versendende Datenstruktur dar, während für *xdr_dummy* diejenige Funktion eingesetzt wird, die den XDR-Filter für die zu übertragende Datenstruktur enthält. Auf der Gegenseite wird als Empfangsroutine die Funktion

int **RecvXDR** *(dsd, ptr, xdr_dummy)*
int dsd;
char ptr;*
*int (*xdr_dummy());*

für die Datenannahme und die Deserialisierung zur Verfügung gestellt. Die Aufrufparameter entsprechen denen der Senderoutine. Die Kommunikationsleistung dieser Routinen hängt neben dem Aufbau der Datenstruktur sehr von der erzeugten Nachrichtenlänge ab. Mit diesen Routinen werden Übertragungwerte bis 800 kByte/sec erreicht. Die Kosten für eine rechnerunabhängige Übertragung und für die damit verbundene Umsetzung der Daten werden deutlich, wenn man diese Routinen mit einfachen Byte-Transfer-Routinen vergleicht. Diese Funktionen erreichen in der gleichen Umgebung 1000 kByte/sec [38]. Aus diesem Grund wurden in dieser Arbeit zwei weitere Routinen hinzugefügt, die speziell Byte-Felder übertragen, wie sie in dieser Arbeit häufig Verwendung finden. Die Senderoutine lautet

int **SendBytes** *(dsd, buf, len)*
int dsd;
char buf;*
int len;

Der erste Parameter gibt den erzeugten Deskriptor an, über den die Kommunikation abläuft. Der zweite Aufrufparameter beinhaltet den Zeiger auf einen Ausgangspuffer, während der letzte Parameter die Anzahl der gewünschten Bytes enthält. Auf der Empfangsseite enthält die Routine

int **RecvBytes** *(dsd, buf, len)*
int dsd;
char buf;*
int len;

die nötigen Funktionen zur Annahme der Byte-Daten. Die Aufrufparameter sind gleich denen der Sendeanweisung. Hier sei ausdrücklich darauf hingewiesen, daß diese Routinen aus Geschwindigkeitsgründen keine Anpassung der Daten vornehmen, die allerdings bei Byte-Feldern auf den hier verwendeten Maschinen nicht nötig waren. Für alle anderen Datentypen sind die Routinen mit externer Datenrepräsentation zur Übertragung verwendet worden.

7.2 Implementiertes Client/Server-Modell

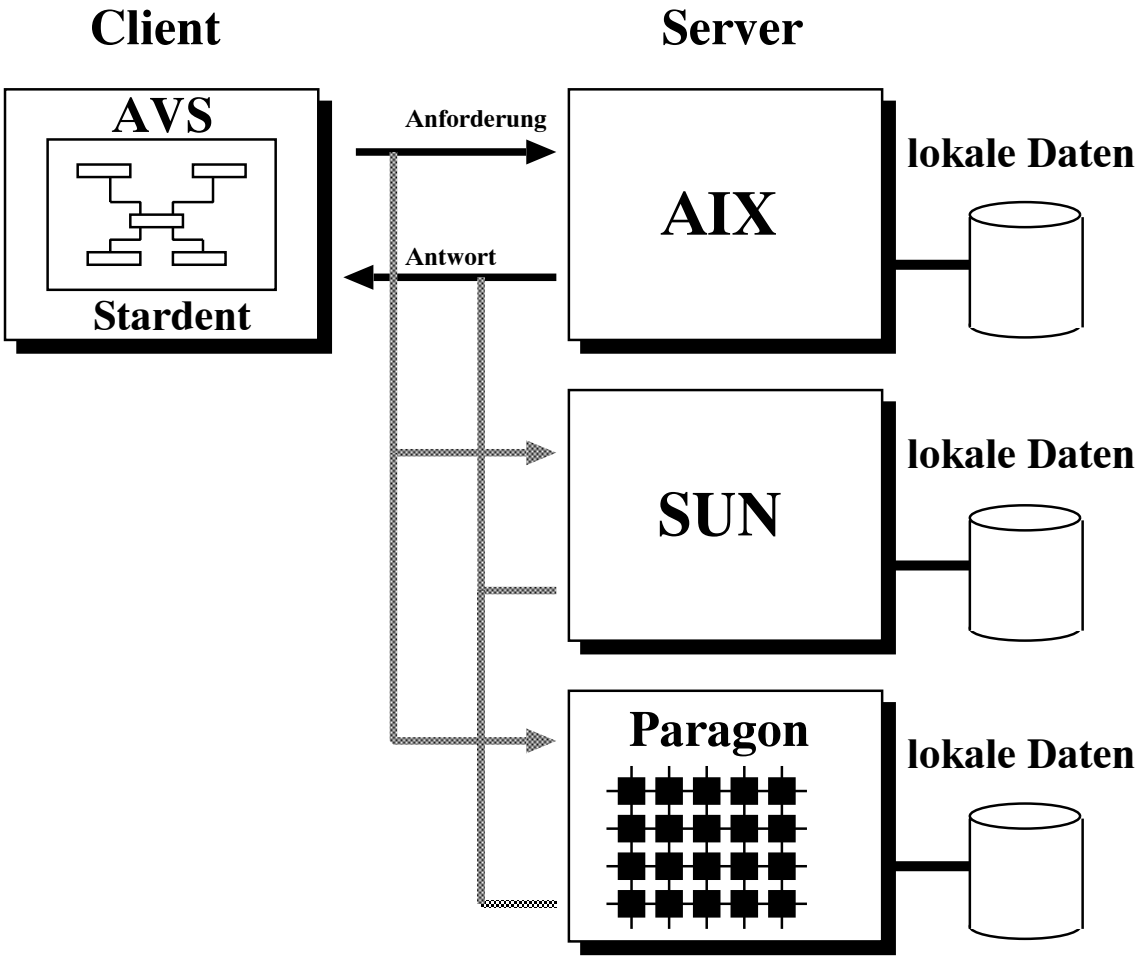


Abbildung 7.3: Übersicht des Client/Server-Ansatz

Wie bereits erwähnt, soll sich die Bedienungsoberfläche des bereits erstellten Systems nicht wesentlich ändern. Somit bleibt die verwendete Umgebung zur Darstellung der Daten weiterhin das AVS-System. Aus diesem Grund ist der Client mit Hilfe des AVS-Systems implementiert. Abb. 7.3 zeigt diese Situation im Überblick. Das hier benutzte AVS-Netzwerk besitzt eine Schnittstelle, die in der Lage ist, einen Auftrag für einen der Server zu erstellen. Es können verschiedene Server gestartet werden, wobei zur Entwicklung der Client/Server-Applikation zunächst sequentielle Versionen auf den Systemen AIX- und SUN-Cluster erzeugt wurden. Der unterste dieser Server stellt den verwendeten Parallelrechner 'Intel Paragon' dar, dessen Algorithmus jedoch erst im folgenden Kapitel betrachtet wird. Der Client formuliert in diesem Modell lediglich den Auftrag, wobei die zu bearbeitenden Daten lokal auf den Servern vorliegen. Als Ergebnis werden die angepaßten Daten an den Client gesendet, der die empfangenen Daten weiterbearbeiten kann.

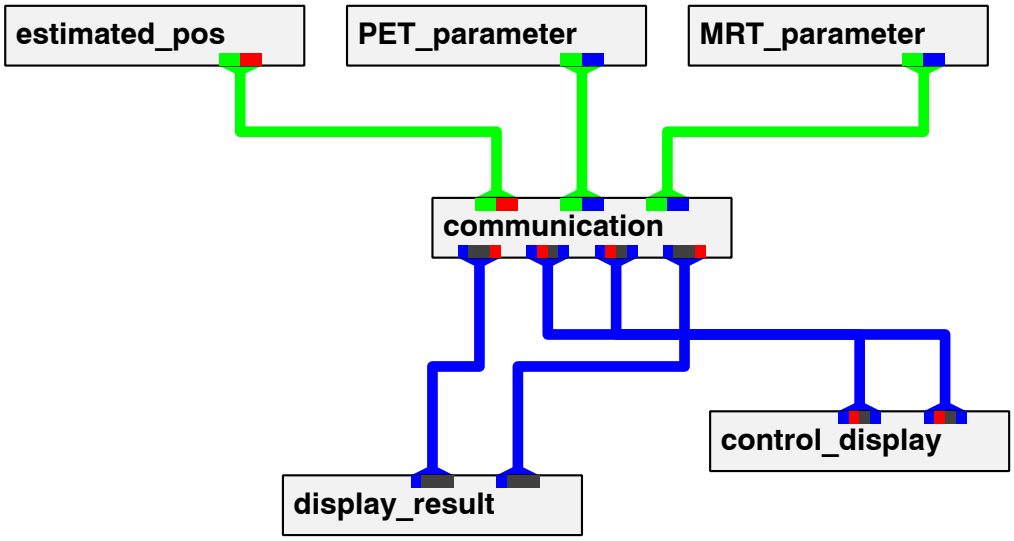


Abbildung 7.4: AVS-Netz für den Client/Server-Ansatz

7.2.1 Client-Implementierung

Im folgenden ist die Struktur des implementierten Client beschrieben. Abb. 7.4 gibt einen Überblick über das für diesen Zweck erstellte AVS-Netzwerk. Verglichen mit der Gesamtübersicht aus Abb. 6.1 fehlen einige wesentliche Teile des sequentiellen Netzwerkes, die hier von einem Server erledigt werden sollen. Die fehlenden Teile des sequentiellen Bearbeitungsnetzes werden durch das Modul *communication* ersetzt, das die Kommunikation zum Server übernimmt und die Daten an die Ausgabemodule weitergibt. Das Kommunikationsmodul erhält von drei Parametermodulen eine Eingabe. Die Module *estimated_pos*, *PET_parameter* und *MRT_parameter* sind identisch mit denen aus der sequentiellen Implementierung und erzeugen entsprechende Datensätze. Die Bedienungselemente dieser Module befinden sich ebenfalls in gleichen Bedienfenstern.

Kommunikationsmodul

Das Kommunikationsmodul hat die Aufgabe, alle für den Server nötigen Parameter zu empfangen und einen Auftrag für den Server zu erstellen. Nach Abschluß der Berechnung sollen die vom Server ermittelten Daten wieder angenommen und an die folgenden Module weitergegeben werden. Der zeitliche Ablauf findet sich in Abb. 7.5, wobei auf dem Server-Rechner das Programm *mrserver* läuft.

Da die Parameter in der sequentiellen Version jedem einzelnen Modul zugeordnet waren und hier alle vom Modul *communication* eingelesen werden müssen, ergibt sich für die Hauptroutine des Moduls der folgende Aufruf:

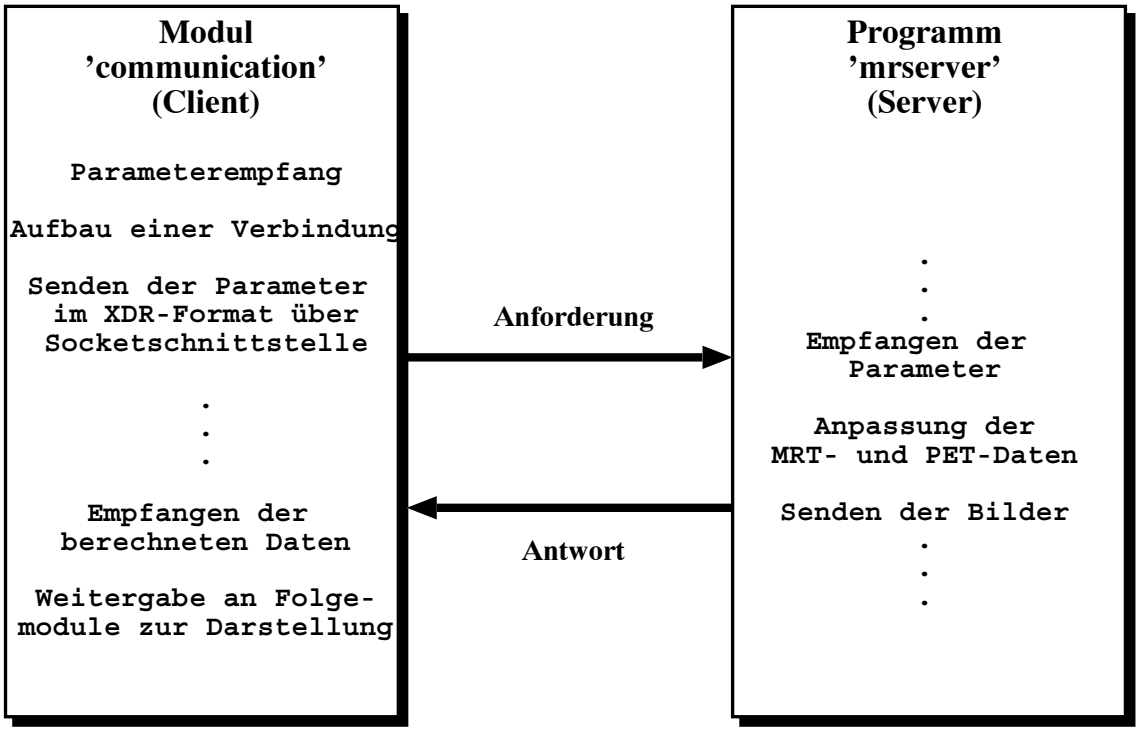


Abbildung 7.5: Zeitlicher Ablauf der Client/Server-Kommunikation

```
int communication_compute( mr_masch_para, pet_masch_para, approx_trans,
mr_slices,mr_control, pet_control, pet_slices, mr_file, pet_file,
pet_level, mr_level, pet_min_len, pet_min_perc, pet_num_of_p,
mr_min_len, mr_min_perc, mr_num_of_p, phi_alpha, phi_beta, sh_alpha,
sh_beta, method, control, pet_red, magx, magy, magz, servername)
        param_type *mr_masch_para;   /* MRT-Maschinenparam. */
        param_type *pet_masch_para;  /* PET-Maschinenparam. */
        transf_type *approx_trans;   /* Startwert */
        AVSfield_char **mr_slices;   /* Ausgabe MRT-Schichten */
        AVSfield_float **mr_control; /* Ausgabe MRT-Punkte */
        AVSfield_float **pet_control;/* Ausgabe PET-Punkte */
        AVSfield_char **pet_slices;  /* Ausgabe PET-Schichten */
        char *mr_file;              /* Name des MRT-Datensatzes */
        char *pet_file;            /* Name des PET-Datensatzes */
        int pet_level;            /* Schwellwert PET */
        int mr_level;            /* Schwellwert MRT */
        int pet_min_len;        /* absolute Min.laenge PET-Kontur */
        float *pet_min_perc;   /* relative Min.laenge PET-Kontur */
        int pet_num_of_p;     /* Anzahl PET-Punkte */
        int mr_min_len;      /* absolute Min.laenge MRT-Kontur */
        float *mr_min_perc; /* relative Min.laenge MRT-Kontur */
        int mr_num_of_p;   /* Anzahl MRT-Punkte */
```

```
        float *phi_alpha;  /* Schritt Differenzb. Winkel */
        float *phi_beta;   /* Schritt Differenzb. Verschiebung */
        float *sh_alpha;   /* Minimale Schrittweite Winkel */
        float *sh_beta;    /* Minimale Schrittweite Verschiebung */
        char *method;      /* gewuenschte Anpassungsmethode */
        char *control;     /* Server-Kontroll-Kommando */
        float *pet_red;    /* Wert fuer Aufloesungskompensation */
        float *magx;       /* Vergroesserung in x-Richtung */
        float *magy;       /* Vergroesserung in y-Richtung */
        float *magz;       /* Vergroesserung in z-Richtung */
        char *servername;  /* Name des Server */
{ ... }
```

Nach den Parametern für die Eingabedaten, folgen die Ausgabedaten, die sich bis auf *mr_control* nicht von den Ausgangsdaten in der sequentiellen Version unterscheiden. Bei *mr_control* wird statt der Konturliste des sequentiellen Falls eine Punkteliste übergeben. Aus diesem Grunde ist das Modul *control_display* leicht geändert; die MRT-Punkterzeugung fällt weg. Die auf die Ausgabedaten folgenden Parameter haben bis auf zwei Ausnahmen die gleiche Bedeutung wie die Gegenstücke aus dem sequentiellen Ansatz. Die beiden Ausnahmen sind die neu hinzugekommenen Parameter *control*, um den Server im Programmablauf zu beeinflussen und *servername*, um die Adresse eines Server im Klartext zu übergeben. Server und Client besitzen eine aufeinander abgestimmte Port-Nummer.

Nach der Übernahme der Parameter werden diese in eine dafür entwickelte Datenstruktur abgelegt. Diese Daten stellen eine Auftrag für den Server dar. Es wird eine Verbindung zum Server aufgebaut und der Auftrag als Anforderung an den Server übermittelt. Der für die Übertragung des Auftrages nötige Anweisungsblock hat das folgende Aussehen:

```
    /* PET-Parameter senden */
    SendXDR(csd,(PET_MR_para_type*)&pet_para,xdr_PET_MR_para_type);
    RecvXDR(csd,(char *)&ackn,xdr_char);   /* Quittung abwarten  */

    /* MRT-Parameter senden */
    SendXDR(csd,(PET_MR_para_type*)&mr_para,xdr_PET_MR_para_type);
    RecvXDR(csd,(char *)&ackn,xdr_char);   /* Quittung abwarten  */

    /* Verfahrensparameter senden */
    SendXDR(csd,(match_type*)&match_para,xdr_match_type);
    RecvXDR(csd,(char *)&ackn,xdr_char);   /* Quittung abwarten  */

    /* Startwert senden */
    SendXDR(csd,(transf_type*)approx_trans,xdr_transf_type);
    RecvXDR(csd,(char *)&ackn,xdr_char);   /* Quittung abwarten  */
```

Diese Übertragung geschieht mit Hilfe der bereits erwähnten externen Datenrepräsentation, in die die Daten umgewandelt werden. Für jeden der hier übertragenen Datentypen wurde ein entsprechender XDR-Filter erstellt. Nach jeder Datenübertragung wird eine kurze Quittung abgewartet, da es sonst vorkommen kann, daß zwei aufeinanderfolgende Nachrichten von den Routinen so schnell an den Empfangsrechner übertragen werden, daß der Empfänger nicht in der Lage ist, sie als zwei Nachrichten zu interpretieren.

Ist der Auftrag vom Server bearbeitet worden, werden die Antwortdaten über den folgenden Anweisungsblock wieder vom Client empfangen:

```
/* Emfangen der PET-Punkte */
RecvXDR(csd,(field_float*)helpfloat,xdr_field_points);
SendXDR(csd,(char*)&ackn,xdr_char); /* Quittung senden */
            ...
/* Empfangen der MRT-Punkte */
RecvXDR(csd,(field_float*)helpfloat,xdr_field_points);
SendXDR(csd,(char*)&ackn,xdr_char); /* Quittung senden */
            ...
/* Empfangen PET-Schichten */
RecvBytes(csd,(*pet_slices)->data,size);
SendXDR(csd,(char*)&ackn,xdr_char); /* Quittung senden */
            ...
/* Empfangen MRT-Schichten */
RecvBytes(csd,(*mr_slices)->data,size);
SendXDR(csd,(char*)&ackn,xdr_char); /* Quittung senden */
```

Die PET- und MRT-Punkte werden zunächst in ein Hilfsfeld übertragen, um dann in das eigentliche Datenfeld umgespeichert zu werden. Im zweiten Schritt werden die PET- und MRT-Schichten als Byte-Felder angenommen. Die Größe dieser Felder kann dabei aus internen Daten des Client bestimmt werden.

Client-Bedienung

Die Client-Bedienung gleicht in weiten Teilen der in Abschnitt 6.5 erläuterten Bedienung des Grundnetzwerkes. Die Fenster *AVS main* und *AVS machine parameter* aus Abb. 6.17 und 6.19 finden sich bei der Client-Implementierung mit der gleichen Funktionalität wieder. Lediglich das Fenster *AVS control* aus Abb. 6.18 hat leichte Änderungen erfahren, wobei die Grundelemente erhalten geblieben sind, siehe Abb. 7.6 . Die beiden ursprünglichen File-Browser sind durch einfache Texteingabezeilen ersetzt worden, die die relative Lage der Daten zum Verzeichnis, aus dem der Server gestartet wird, enthalten. Zu den bereits vorhandenen Feldern ist das Feld *Hosts* hinzugetreten. Mit Hilfe der oberen Auswahlknöpfe läßt sich einer der dort angegebenen Rechner anwählen, wobei es dem Client möglich sein muß, diesen Namen der entsprechenden Maschine zuzuordnen. Die untere Knopfleiste

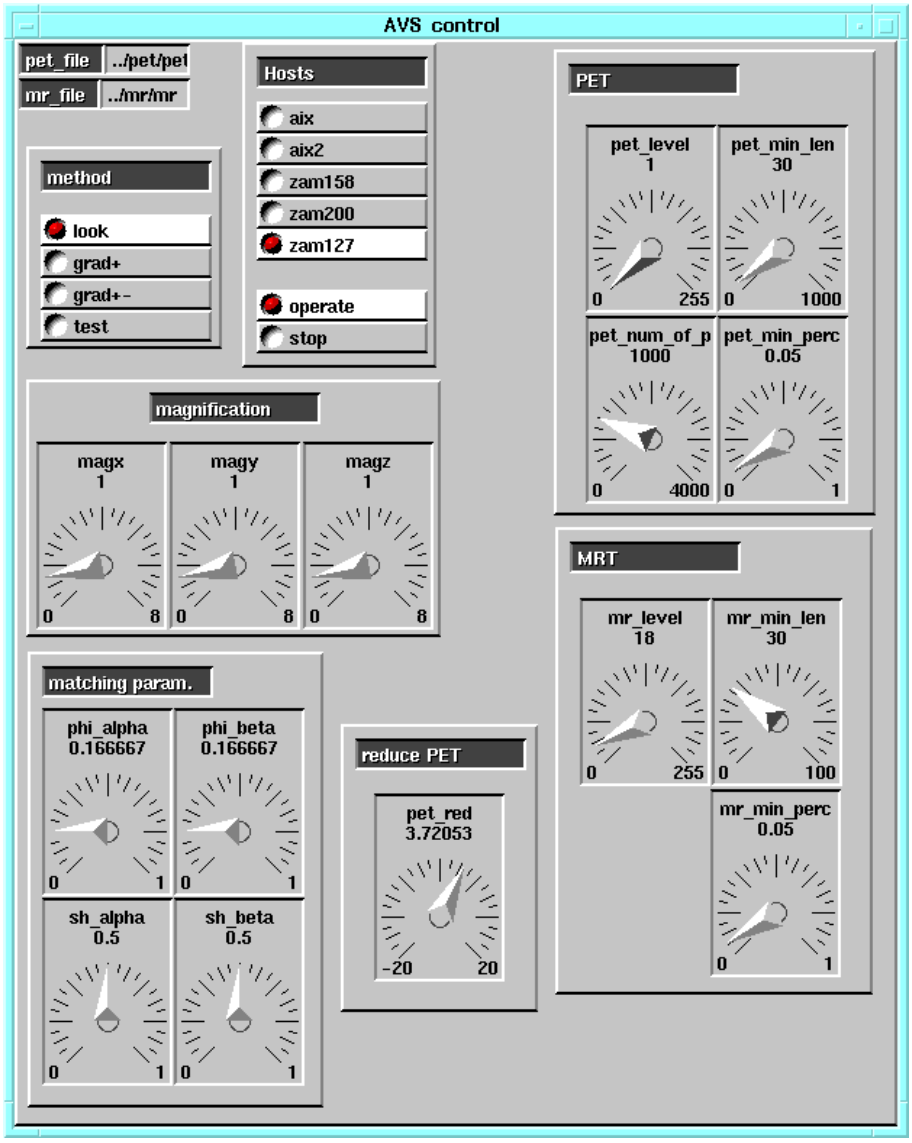


Abbildung 7.6: Client-Bedienfeld

enthält zwei Kommandos, die dem Server übergeben werden können. Bei *operate* arbeitet der Server nach der Auftragsbearbeitung weiter und wartet auf einen neuen Client-Auftrag, während *stop* den Server nach dessen Durchlauf terminiert.

7.2.2 Server-Implementierung

In diesem Abschnitt wird die Implementierung des Server betrachtet. Es handelt sich zunächst um eine sequentielle Server-Version, die unabhängig von AVS ist.

Das Server-Programm ist unabhängig von AVS geschrieben und ist damit frei von speziellen für AVS notwendigen Bibliotheken. Desweiteren entfällt eine Kontrolle des Server durch ein übergeordnetes System wie AVS. Somit ist man nicht mehr an

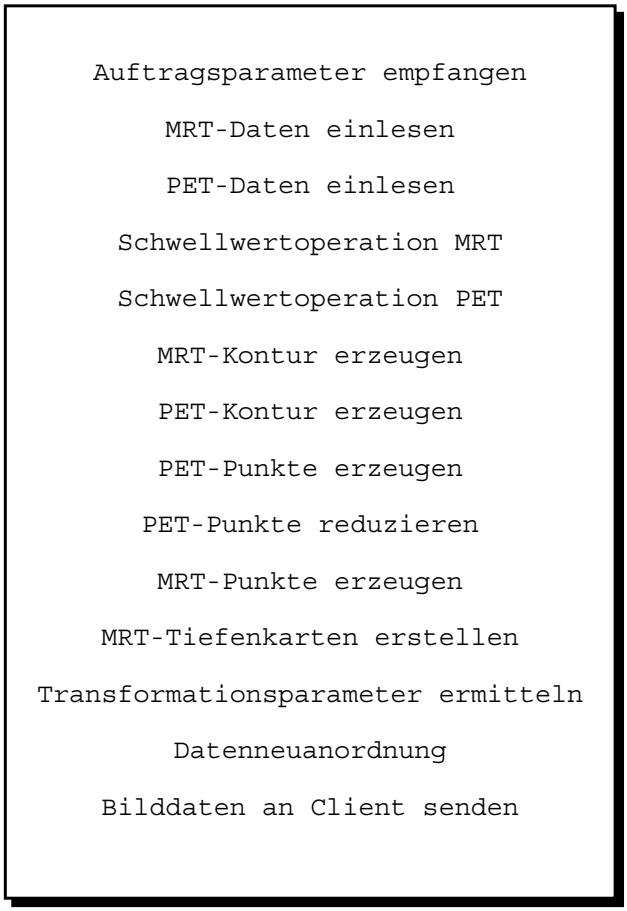


Abbildung 7.7: Zeitlicher Ablauf im Server

die starre Modulstruktur des AVS gebunden und kann selbst Ablaufstrategien und Datenübergaben steuern. Der zeitliche Ablauf innerhalb des Server ist in Abb. 7.7 dargestellt. Die einzelnen Teilschritte gleichen den Modulen des sequentiellen Algorithmus unter AVS. Die Module sind im Server als externe Unterprogramme bekannt, wobei die Berechnungsroutinen den ursprünglichen AVS-Modulen entnommen und entsprechend der neuen Umgebung angepaßt wurden. Die AVS-Datenstrukturen wurden hierbei durch vergleichbare eigene Strukturen ersetzt. Für den Ablauf des Algorithmus ist ein Hauptprogramm erstellt worden, das alle benötigten Module in geeigneter Weise anspricht. Die in den ursprünglichen Modulen enthaltenen Speicherverwaltungsroutinen wurden entfernt und diese Aufgabe dem Hauptprogramm des Server übertragen, da so die Speicherverwaltung unabhängig von den Untermodulen zentral gesteuert werden kann.

Der gesamte Anweisungsblock in Abb. 7.7 wird so oft durchlaufen, bis der Server das Kommando *stop*, vgl. Abschnitt 7.2.1, bekommt. Nach jedem Durchlauf werden die Parameter des alten Durchlaufs abgespeichert und stehen beim nächsten Auftrag zur Verfügung. Hiermit ist es möglich, nur diejenigen Module anzusprechen, die für einen neuen Client-Auftrag durchfahren werden müssen. Neben Änderungen in den Parametern spielt hier auch der Datenfluß eine entscheidende Rolle.

Dieses Verfahren entspricht der AVS-Funktionalität, daß Module nur dann neu angestoßen werden, wenn sich die Eingangsdaten oder Parameter ändern. Innerhalb eines Teilanweisungsblocks des Hauptprogramms findet neben dem Aufruf der eigentlichen Berechnungsroutine, die Speicherverwaltung für das aufzurufende Modul statt. Zur Kontrolle des Server wird eine Bestätigung nach jedem Teilanweisungsblock ausgegeben.

Die Initialisierung des Server umfaßt neben der Bereitstellung der grundlegenden Datenstrukturen die Schaffung der Server-Schnittstelle für einen möglichen Client. Nach der Initialisierung ist der Server bereit, Aufträge entgegenzunehmen. Die für den Parameterempfang bereitgehaltenen Anweisungen übernehmen die Daten aus dem Client entsprechend der in Abschnitt 7.2.1 gezeigten Sendeschnittstelle des Client. Die Ausgangsdaten liegen lokal auf dem Server vor, wobei den Daten kein Header wie in der Version unter AVS vorangestellt wurde. Die Daten sind hier als Byte-Felder abgelegt. Für diesen Zweck ist für die Server-Version ein eigenes Dateneingabemodul geschrieben worden, das die MRT- und PET-Datensätze einliest. Wie bereits erwähnt entsprechen die auf die Dateneingabe folgenden Teilanweisungsblöcke bis auf die Anweisungen für die Bilddatenübertragung den entsprechenden Modulen der Version unter AVS, siehe 6.1. Die Übertragung der Ergebnisdaten des letzten Blockes entspricht der in Abschnitt 7.2.1 gezeigten Datenempfangsroutine.

Die Portierung des Server auf andere UNIX-Systeme wird durch die Nutzung vorhandener Standard-Bibliotheken vereinfacht. Sollen dagegen unter AVS vorhandene Versionen portiert werden, so muß auf ein auf dem Zielrechner vorhandenes AVS-System zurückgegriffen werden. Der hier gezeigte Server ermöglicht es, neben dem bereits gezeigten Client, auch anders implementierte Clients zuzulassen, die sich dann jeweils nur an die Schnittstellendefinition des Server halten müssen. Dies kann sinnvoll sein, wenn ein Client aufgebaut werden soll, der mit einer anderen Visualisierungs-Software als AVS ausgestattet ist.

Kapitel 8

Paralleler Ansatz

In diesem Kapitel wird die parallele Version des verwendeten Anpassungsalgorithmus beschrieben und bewertet. Nach einer kurzen Erläuterung zu dem verwendeten Rechnersystem Paragon der Firma Intel erfolgt die Betrachtung des parallelen Verfahrens.

8.1 Das System Intel Paragon

Hier soll nur ein kurzer Überblick über das verwendete Rechnersystem gegeben werden. Für weitere Informationen sei auf [9] verwiesen.

8.1.1 Systemcharakteristik

Der verwendete Rechner Intel Paragon XP/S 10 ist ein massiv-paralleles System, das im Forschungszentrum Jülich derzeit mit 140 Rechenknoten installiert ist. Es handelt sich um eine Architektur mit verteiltem Speicher. Jeder Knoten besitzt einen lokalen Speicher von 32 Mbyte sowie eine Applikationsprozessor vom Typ i860 XP, der mit 50 MHz getaktet wird. Über ein intelligentes Netzwerk-Interface auf den Knoten ist jeder Knoten mit einem speziellen iMRC-Baustein verbunden. Neben der Verbindung zu einem Knoten unterhält dieser Baustein Verbindungen zu vier weiteren iMRC. Auf diese Weise entsteht eine Gitterstruktur, die alle Knoten untereinander verbindet und das Verbindungsnetzwerk bildet. Neben den Rechenknoten existieren dazu baugleiche I/O-Knoten, die die Ein- und Ausgabe des Systems steuern. In dem verwendeten System befinden sich 7 dieser Knoten, die die Verbindung zu 6 Festplattensystemen und die Schnittstellen nach außen (z.B. Ethernet, V24, FDDI) bedienen. Weitere 4 Knoten sind nur für die Systemunterstützung wie Zugang zum System und Programmentwicklung, vorgesehen und können nicht als Rechenknoten für eine parallele Applikation genutzt werden.

Auf jedem Knoten läuft als Betriebssystem 'Paragon OSF/1', das auf dem UNIX-Betriebssystem OSF/1 [37] basiert. Paragon OSF/1 enthält alle Standardfunktionen des OSF/1, erweitert durch eine Reihe von speziellen Funktionen für das hier beschriebene parallele System [28][9]. Aufgrund dieser speziellen Funktionen erscheint das gesamte System dem Benutzer als ein Ein-Prozessor-UNIX-System, wie er es von einer Workstation her kennt.

Wie bereits erwähnt, übernehmen die Knoten zum Teil sehr unterschiedliche Aufgaben. Aus diesem Grunde werden Knoten mit gleichen Aufgaben in einer hierarchischen Struktur von Partitionen zusammengefaßt. Die Service-Partition enthält alle Knoten, die der Systemunterstützung dienen, während die Berechnungs-Partition alle Berechnungsknoten enthält. Teile dieser Partition werden als Benutzer-Partitionen den Benutzern für ihre parallelen Applikationen zur Verfügung gestellt. Die I/O-Knoten bilden die Ein-/Ausgabe-Partition, werden aber meist der Service-Partition zugeordnet [9]. Alle Partitionen werden in der Root-Partition zusammengefaßt.

Die Programmerstellung kann auf dem System selber oder einer Workstation erfolgen, die in der Lage ist, ausführbare Programme für das Paragon System zu erstellen. Als Compiler wurde im Rahmen dieser Arbeit der mitgelieferte C-Compiler 'icc' verwendet. Dieser Compiler besitzt eine Reihe von Optionen, die ausführlich in [26] beschrieben werden.

Für das hier beschriebene System werden unterschiedliche Programmiermodelle angeboten [9], wobei hier das im folgende näher erläuterte Message Passing verwendet wurde.

8.1.2 Message Passing

Bei dem hier betrachteten Programmierkonzept Message Passing wird auf jedem an der parallelen Applikation beteiligten Rechenknoten das gleiche Programm als Prozeß gestartet. Die einzelnen Knoten können untereinander Nachrichten austauschen, die jeweils einen Adressaten und einen Datenblock besitzen. Auf diese Weise findet der Datenaustausch sowie die Synchronisation der Knoten untereinander statt.

Auf dem System Paragon ist hierzu eine Reihe von Funktionen zum Nachrichtenaustausch implementiert [27], wobei hier kurz die verwendeten Routinen in ihren Möglichkeiten dargestellt werden sollen. Eine Bestimmung der Kommunikationsleistung des Systems findet sich in [14].

Man unterscheidet zwischen blockierender und nicht-blockierender Kommunikation. Bei der blockierenden Kommunikation wartet ein Prozeß nach dem Aufruf der Kommunikationsroutine auf dessen Ausführung, während bei der nicht-blockierenden Kommunikation der Prozeß nur den Kommunikationsauftrag an das Netzwerk-Interface weitergibt, aber nicht auf eine Ausführung der Kommunikation wartet. Mit den letztgenannten Operationen ist es möglich, gleichzeitig Daten senden bzw.

empfangen zu lassen und weiter an einer Berechnung zu arbeiten. Aufgrund der dann gleichzeitig aus dem Speicher ablaufenden Kommunikation muß vor dem Zugriff auf den Sende-/Empfangspuffer geprüft werden, ob eine Nachricht bereits vollständig empfangen oder gesendet ist.

Als Routine zum blockierenden Senden wird

void **csend** *(type, buf, len, node, pid)*

verwendet, während

void **crecv** *(type, buf, len)*

für den blockierenden Empfang einer Nachricht Verwendung findet. *type* kennzeichnet hierbei eine vom Benutzer frei wählbare Nachrichtenkennung. Kennungen können kombiniert werden, um beispielsweise Nachrichten mit verschiedenen Kennungen zu empfangen. Hat *type* den Wert -1, werden Nachrichten mit beliebiger Kennung empfangen, weitere Beispiele finden sich in [27]. Mit *buf* wird der Zeiger auf den zu sendenden bzw. empfangenden Datenbereich übergeben, während *len* die Länge dieses Puffers kennzeichnet. Bei der Sendeanweisung finden sich desweiteren die beiden Parameter *node* und *pid*, die die Knotennummer und die Prozeßnummer des Adressaten angeben. Wird für *node* der Wert -1 angegeben, so wird an alle beteiligten Knoten die zu sendende Nachricht geschickt. Dieser Fall wird 'Broadcast' genannt. Die Routinen *isend* und *irecv* werden in analoger Weise für eine nichtblockierende Kommunikation genutzt. Soweit nicht anders vermerkt, wird in dieser Arbeit die blockierende Kommunikation verwendet.

Häufig ist man nicht an einer konkreten Nachricht von einem Knoten zu einem anderen interessiert, sondern möchte eine Operation global über alle Knoten durchführen. Als Beispiel sei hier die Funktion

void **gdhigh** *(x, n, work)*

genannt. Diese Funktion ermittelt das globale Maximum für jedes Element in einem Wertefeld x, das lokal auf jedem Knoten vorhanden ist. In dem Aufruf gibt n die Länge des Feldes an. Der Parameter *work* ist ein Zeiger auf ein Hilfsfeld, das mindestens die Länge von x haben muß. Die gezeigte Funktion ist für Fließpunktwerte mit doppelter Genauigkeit gedacht, existiert aber in gleicher Form auch für Ganzzahlwerte und Fließpunktwerte mit einfacher Genauigkeit. Die entsprechenden Routinen heißen dann *gihigh* und *gshigh*. Neben der Bestimmung des Maximums existieren weitere Funktionen zur Bestimmung des Minimums (*gdlow*), zur Berechnung eines globalen Produkts (*gdprod*), einer Summe (*gdsum*) und für Ganzzahlwerte existieren logische Funktionen (*giand,gior*). Neben den vordefinierten Funktionen gibt es eine allgemeine Funktion *gopf*, die die globale Kommunikation übernimmt, der aber eine Funktion übergeben werden muß, die global ausgeführt werden soll. Wesen der

globalen Operationen ist, daß alle Knoten der Anwendung diesen Befehl gleichzeitig ausführen müssen, so daß eine globale Synchronisation stattfindet. Will man nur eine globale Synchronisation erreichen, ohne Benutzernachrichten auszutauschen, steht hierfür die Funktion *gsync* zur Verfügung. Neben einer globalen Funktionsberechnung existieren weitere Funktionen, die Daten von allen Knoten sammeln und sie allen Knoten wieder zur Verfügung stellen. Eine solche Operation ist:

void **gcol** *(x, xlen, y, ylen, *n)*

Hierbei stellt jeder Knoten den lokalen Puffer x mit der lokalen Länge *xlen* zur Verfügung. Diese lokalen Puffer werden entsprechend der Knotennummer hintereinander eingesammelt und anschließend jedem Knoten über den Ergebnispuffers y der Länge *ylen* zur Verfügung gestellt. Während *ylen* die vom Benutzer vorgegebene Länge des Empfangspuffers ist, gibt n an, wieviele Werte von der Funktion tatsächlich eingesammelt wurden.

Für die Implementierung von parallelen Programmen auf mehreren Knoten ist häufig die Information wichtig, wieviele Knoten an der Applikation beteiligt sind, und welche Knotennummer der gerade betrachtete Knoten besitzt. Für diese beiden Operationen existieren die Funktionen *mynode* und *numnodes*, wobei *mynode* die Knotennummer und *numnodes* die Gesamtzahl der Knoten einer Applikation zurückgibt.

8.2 Parallele Implementierung

Im weiteren wird die parallele Implementierung des in Abschnitt 7.2.2 beschriebenen sequentiellen Client/Server-Programms näher erläutert. Die Server-Schnittstelle der parallelen Implementierung wurde gegenüber der sequentiellen Version nicht geändert. Da der Benutzer diesen Server über die in Abschnitt 7.2.1 angegebene Bedienungsoberfläche anspricht, bleibt ihm die konkrete Implementierung des Server verborgen. Hier wird zunächst eine Gesamtübersicht über die parallele Version gegeben, der sich eine genauere Betrachtung der Teilalgorithmen anschließt.

8.2.1 Gesamtüberblick

Für die Parallelisierung eines Programms muß der ursprüngliche Algorithmus auf Teile untersucht werden, die sich parallel ausführen lassen. Dabei kann man vorhandene Daten- oder Funktionsparallelität nutzen. In diesem Fall bietet sich eine Ausnutzung der Datenparallelität an, da die Daten als zunächst voneinander unabhängige Schichtbilder vorliegen. Neben der Ausnutzung paralleler Strukturen hat das verwendete Programmierkonzept des Message Passing einen entscheidenden Einfluß auf die Implementierung.

Datenverteilung

Der hier gewählte Ansatz zur Datenverteilung ist, die vorhandenen Schichten auf die Zahl der benutzten Knoten zu verteilen. Hierbei ist es wichtig, eine eindeutige Zuordnung von Schichten und Prozessoren zu finden. Um eine gute Lastverteilung zu erreichen, sollen hier die Schichten möglichst gleichmäßig auf die Knoten verteilt werden. Ein Ansatz, die Daten weiter zu zergliedern, wird hier nicht verfolgt.

Für die Datenaufteilung werden jeweils die Anzahl und die erste zu bearbeitende Schicht bestimmt. Die erste Schicht $s_{first}(i)$ die einem Knoten i $(i = 0, 1, \ldots, n-1)$ zugeordnet wird, berechnet sich durch:

$$s_{first}(i) = \frac{i}{n} \cdot N, \tag{8.1}$$

wobei die Nachkommastellen abgeschnitten werden. Die zu bearbeitende Anzahl $s_{number}(i)$ wird durch

$$s_{number}(i) = s_{first}(i+1) - s_{first}(i) \tag{8.2}$$

berechnet. Hierbei ist n die Gesamtanzahl der verwendeten Knoten und N die Gesamtzahl der Schichten, wobei die Numerierung bei 0 beginnt. Nach Gl. 8.1 und Gl. 8.2 werden dem Knoten i die Schichten mit den Nummern $s_{first}(i)$ bis $s_{first}(i+1) - 1$ zugeordnet. Diese Technik wird sowohl für die MRT- als auch für die PET-Schichten angewandt. Abb. 8.1 zeigt im Überblick die Verteilung der Schichten auf n Knoten. Die linke Seite zeigt hierbei die MRT-Daten, während auf der gegenüberliegenden Seite die PET-Daten dargestellt sind.

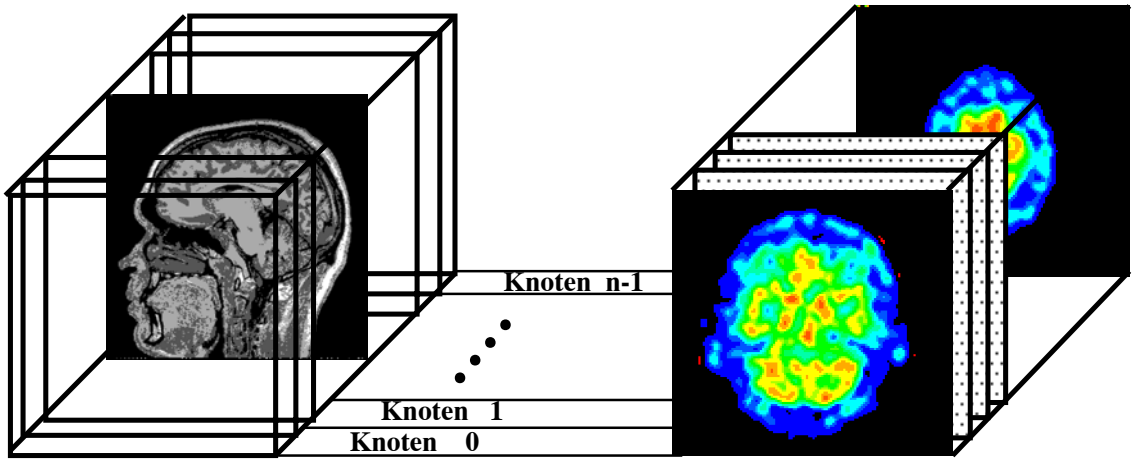


Abbildung 8.1: Datenverteilung auf dem parallelen System

Ist die Zahl der Knoten größer als die Zahl der Schichten, so erhalten einige Prozessoren keine zu bearbeitende Schicht, während alle anderen eine Schicht erhalten. Mit dieser Methode werden die Schichten gleichmäßig auf die Knoten verteilt, wobei sich die Schichtanzahlen pro Knoten um maximal eins unterscheiden.

Gesamtaufbau

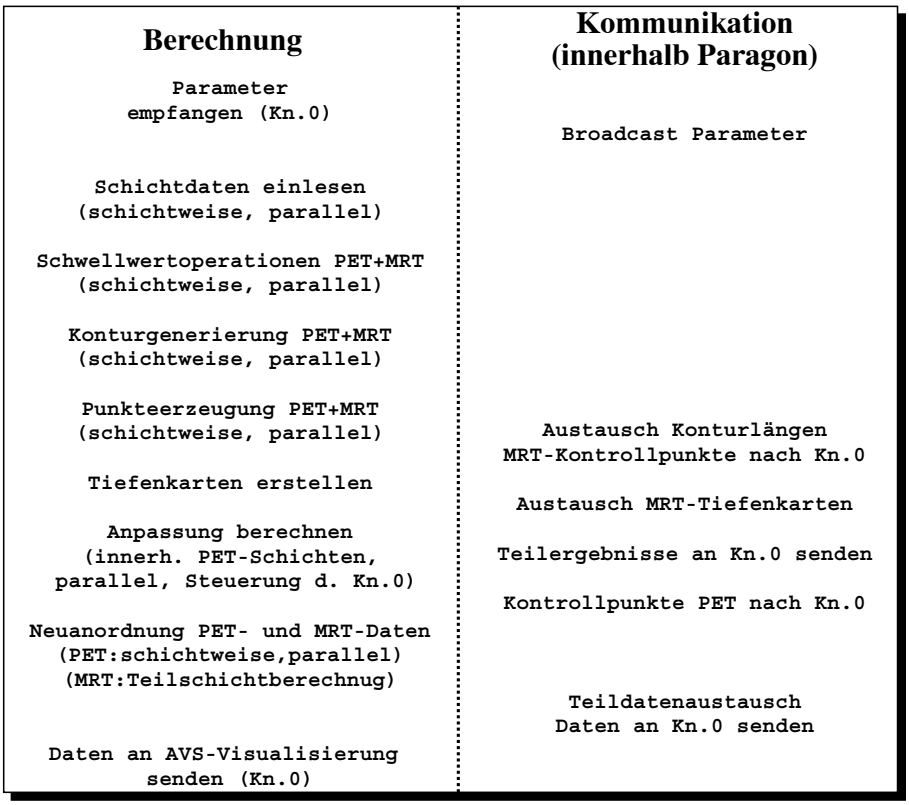


Abbildung 8.2: Zeitlicher Ablauf des parallelen Server

In Abb. 8.2 ist der zeitliche Ablauf des parallelisierten Server-Programms dargestellt, der deutliche Ähnlichkeit mit dem des Server-Programms aus Abschnitt 7.2.2 zeigt. Hier wird eine Übersicht des Gesamtprogrammes geben, während in Abschnitt 8.2.2 die einzelnen Teilalgorithmen näher untersucht und erläutert werden. Während des ganzen Programmablaufs hat der Knoten 0 eine gesonderte Stellung, da globale Entscheidungen, die die gesamte parallele Applikation betreffen, auf diesem Knoten getroffen werden. Da der Client sich nicht mit allen Knoten gleichzeitig verbinden kann, übernimmt Knoten 0 auch die Aufgabe der externen Datenkommunikation zum Client. In der Abb. 8.2 sind die Aufgaben nach Berechnung und externer Kommunikation auf der einen Seite und lokaler Kommunikation zwischen den Knoten auf der anderen Seite unterteilt.

Nach dem Empfang der Parameter auf Knoten 0 werden diese an alle übrigen Knoten durch ein Broadcast verschickt. Danach lesen alle Knoten die ihnen zugeordneten Schichten als Dateien ein. Hierbei werden alle Knoten gleichzeitig aktiv. Die Schwellwertoperationen sowie die Konturgenerierung der Daten erfolgt innerhalb der Schichten parallel, wobei keine Kommunikation der Knoten untereinander nötig ist. Die Punkteerzeugung findet weitgehend mit lokalen Daten statt, wobei die Länge der Gesamtkontur ausgetauscht wird. Der Austausch ist nötig, um bei jeder Knotenzahl das gleiche numerische Ergebnis für die Anpassung zu erhalten. In diesem

Schritt werden die für die Kontrollausgabe nötigen MRT-Punkte erzeugt und auf Knoten 0 gesammelt. Anschließend berechnen die Knoten die für das Verfahren nötigen Tiefenkarten, wobei jeder Prozessor nur den Teil der Tiefenkarte erzeugen kann, dessen Daten er besitzt. Aus diesem Grund muß im Anschluß an diese Operation eine globale Kommunikation stattfinden, die die einzelnen Teilkarten zusammenfügt und wieder an alle Knoten verteilt. Sind die Karten verteilt, kann die Berechnung der Anpassung einsetzen, bei der der Abstand der auf jedem Knoten lokal vorhandenen Punkte zur MRT-Oberfläche minimiert wird. Da die PET-Kontur als Ganzes betrachtet werden muß, wird nur die Berechnung des Abstandes parallel ausgeführt, wobei das Ergebnis dem Knoten 0 bekannt gemacht wird. Dieser Knoten ermittelt mit diesen Ergebnissen neue Transformationsparameter und teilt sie den übrigen Knoten mit. Dies gilt auch für das Endergebnis. Nach Abschluß der Anpassung berechnet jeder Knoten die neuen Positionen der lokal vorhandenen PET-Punkte und sendet diese an den Knoten 0. Mit den erzeugten Transformationsparametern wird im folgenden Schritt eine Neuanordnung der Daten berechnet, wobei die PET-Schichten lokal in den jeweiligen Knoten erzeugt werden können. Die den PET-Schichten zuzuordnenden MRT-Daten werden, soweit auf den Knoten vorhanden, den jeweiligen Schichten zugeordnet. Nach der Neuberechnung werden die ermittelten Teilschichten auf dem Knoten 0 gesammelt und an den Client übergeben.

Meßverfahren

Die hier folgenden Zeitmessungen zeigen den Gewinn durch eine Parallelisierung des Server-Programms. Bevor die Ergebnisse der Zeitmessungen präsentiert werden, werden in diesem Abschnitt einige Bemerkungen zum Meßverfahren gemacht.

Jede der einzelnen Teiloperationen wurde für sich gemessen. Hierzu fand vor der entsprechenden parallelen Routine eine globale Synchronisation (*gsync*) statt. Danach wurde die Startzeit mit *dclock* festgehalten und der Unterprogrammteil abgearbeitet. Am Ende einer Routine wurde auf jedem Knoten die Endzeit festgehalten (*dclock*) und auf eine globale Synchronisation gewartet. Nach der Synchronisation wurde die lokale Zeitdifferenz bestimmt und die Zeit für die *dclock*-Operation abgezogen. Da die Prozessoren teilweise unterschiedliche Rechenzeiten benötigen, wurde anschließend das globale Maximum der benötigten Zeit bestimmt. Es wurden mehrere Reihen von Messungen durchgeführt, die bei allen Ein-/Ausgabeoperationen starke Schwankungen zeigten. Sie beruhen auf Interferenzen mit anderen Benutzern, die sich gleichzeitig auf der Maschine befanden. Für Knotenzahlen kleiner als 80 wurde auf eine dedizierte Messung verzichtet, um den allgemeinen Betrieb auf der Maschine nicht unnötig zu stören. Aufgrund der Meßwertschwankungen sind für die Ein-/Ausgabeoperationen die kleinsten Zeiten genommen worden. Bei anderen Operationen traten sehr stabile Ergebnisse auf, so daß dort ein Mittelwert aus den besten fünf Messungen verwendet wurde.

Das Programm für die Messung ist mit dem C-Compiler 'icc-Version 4.1.1' über-

setzt worden und lief mit der Betriebssystemversion 'Paragon Operating System Release 1.1'. Die verwendeten Compiler-Optionen waren '-O4', '-Mvect' und '-nx', siehe [26]. Nach Starten des Server wurden mehrere Client-Aufträge bearbeitet, ohne den Server zu beenden. Dies entspricht mehr der Nutzung eines Server, als der jeweilige Neustart des Programms. Da die Aufträge jedoch jeweils identisch waren, wurde der Server dazu veranlaßt, bei jedem Auftrag die gesamte Bearbeitungskette zu starten. Normalerweise würde er nur Daten berechnen, die einer Aktualisierung bedürfen.

Da für das System Paragon keine sequentielle Version vorhanden war, wurden der Speedup-Wert für das Programm auf einem Knoten auf 1 gesetzt. Die Messungen fanden jeweils mit folgenden Prozessorzahlen statt: $1, 2, \ldots, 18, 20, 25, 32, 35, 40, 50, 60, 64, 80, 100, 120, 128, 140$. Da für die Messungen die gleichen Daten wie bei der sequentiellen Implementierung verwendet wurden, ist die Problemgröße hier entsprechend, vgl. Abschnitt 6.2 .

Meßergebnisse Gesamtalgorithmus

Die in diesem Abschnitt präsentierten Meßwerte geben einen Überblick des zeitlichen Server-Verhaltens. Vor diesem Hintergrund lassen sich die im weiteren untersuchten Teilalgorithmen hinsichtlich ihres Beitrags zum Gesamtprogramm besser beurteilen.

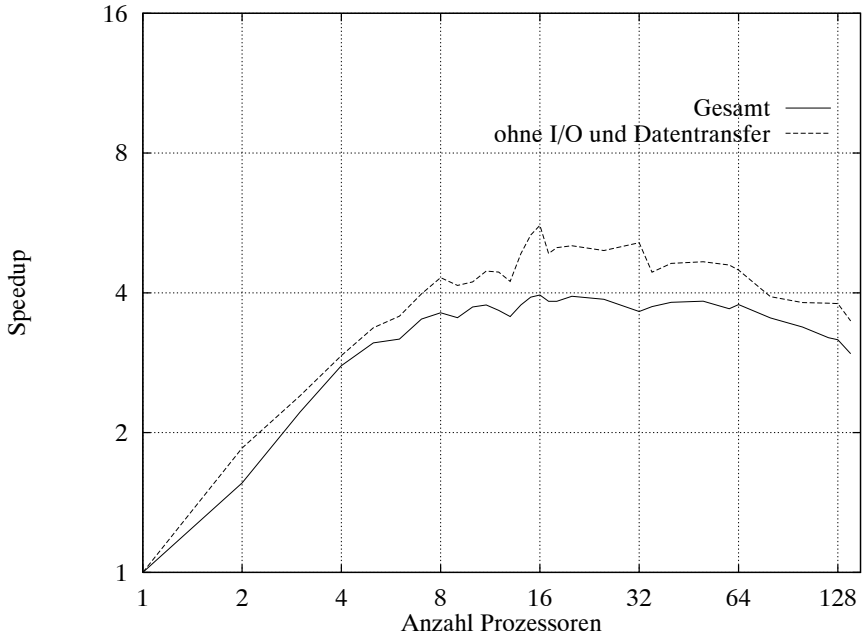


Abbildung 8.3: Speedup des Gesamtverfahrens

In Abb. 8.3 sind die erreichten Speedup-Werte des gesamten Verfahrens dargestellt. Sowohl Speedup als auch Prozessorzahl sind logarithmisch aufgetragen. Bei der Berechnung des Gesamt-Speedup werden auch die Dateieingabe und die Kommunikation mit dem Client berücksichtigt. Der maximale Speedup von vier wird bei einer Anzahl von 16 Prozessoren erreicht. Abb. 8.4 zeigt eine Aufschlüsselung der

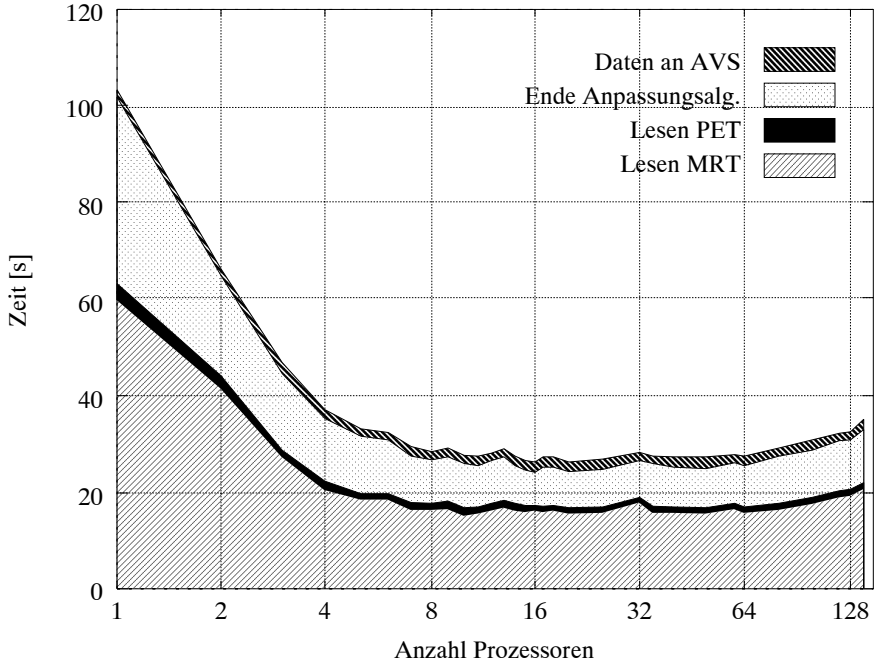


Abbildung 8.4: Zeitübersicht des Gesamtverfahrens

für den Gesamtalgorithmus benötigten Zeiten. Die gekennzeichneten Bereiche geben jeweils die für eine bestimmte Operation benötigte Zeit an. Während anfangs die Gesamtzeit stark abfällt, sinkt sie zwischen 8 und 64 Prozessoren kaum noch. Ab 64 Prozessoren steigt die Gesamtzeit wieder leicht an. Der Anteil Ein-/Ausgabe zu der Zeit für den eigentlichen Algorithmus bewegt sich bei etwa 65% und erreicht bei 16 Prozessoren sein Maximum von 75% um anschließend wieder leicht zu sinken. Die Übertragung der Enddaten an das AVS-System benötigt immer die gleiche Zeit, da die Daten nur sequentiell über das Netz übertragen werden. Währenddessen erreichen die Leseoperationen eine deutliche Zeitersparnis, die im Rahmen der Teilalgorithmen weiter diskutiert wird.

Da neben der Beschleunigung des Gesamtverfahrens auch der Speedup des eigentliche Anpassungsalgorithmus interessant ist, zeigt Abb. 8.3 zusätzlich den Speedup, wie er für diesen Algorithmus ohne die zugehörigen Ein-/Ausgabeoperationen erreicht wird. Hierbei wird der maximale Speedup bei ebenfalls 16 Prozessoren erreicht. Abb. 8.5 zeigt im Überblick die Zeiten, die für die jeweiligen Teiloperationen benötigt werden. Die benötigten Zeiten für die Schwellwert- und Konturoperationen sinken sehr deutlich und verlieren gegenüber anderen Operationen immer mehr an Bedeutung. Sie skalieren sehr gut, wie auch in der detaillierten Betrachtung zu sehen sein wird. Die Daten für MRT und PET wurden hier zusammengefaßt, wobei die PET-Operationen einen nur sehr kleinen Zeitanteil von unter 10% an der zusammengefaßten Zeit hatten. Der Grund hierfür ist die sehr unterschiedliche Größe der Datensätze. Die Ausführungszeit für die Erzeugung der Tiefenkarten und Punkte steigt mit zunehmender Prozessorzahl. Die Punkteerzeugung wurde hier mit der Tiefenkartenerzeugung zusammengefaßt, hat aber nur einen geringen Anteil

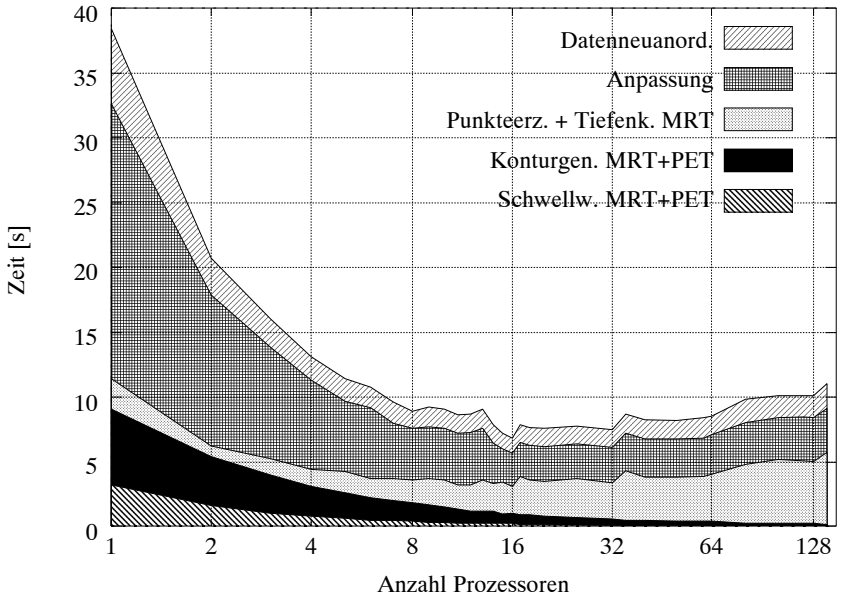


Abbildung 8.5: Zeitübersicht des Gesamtalgorithmus ohne Datenein- und ausgabe

von anfangs 20% der dann auf unter 3% der zusammengefaßten Zeit zurückgeht. Dies zeigt besonders deutlich, wie der nötige globale Austausch der Tiefenkarten das Zeitverhalten für diese Operation beeinflußt. Die Zeit für die eigentliche Anpassungsberechnung sinkt von anfangs $22s$ auf etwa $2{,}5s$, die bei 15 und 16 Prozessoren erreicht werden. Hierbei spiegelt sich wiederum die Anzahl der beteiligten PET-Schichten wider, wie später noch weiter erläutert wird. Der Zeitverbrauch für die Datenneuanordnung sinkt von $5s$ auf etwa $1{,}2s$ ab, wobei mit weiter zunehmender Prozessorzahl ein leichter Wiederanstieg zu beobachten ist.

8.2.2 Teilalgorithmen

Nach der Betrachtung der einzelnen Algorithmen in der Übersicht findet hier eine detailliertere Analyse der Einzelalgorithmen statt. Neben den erreichten Beschleunigungsfaktoren wird ein Einblick in die jeweilige Parallelisierung der Teilalgorithmen gegeben. Dieser Einblick dient zugleich dem Verständnis für die auftretenden Verläufe. Die Client-Kommunikation wird hier außer Acht gelassen, da die Datenübertragung immer sequentiell von Knoten 0 aus erfolgt.

Dateneinlesen

Im Gegensatz zu den Gesamtdatensätzen der sequentiellen Version, die alle Schichten beinhalten, wird hier jede einzelne Schicht als einzelnes File abgelegt und mit einer Nummer im Dateinamen versehen. Über diesen Namen kann jeder Knoten die

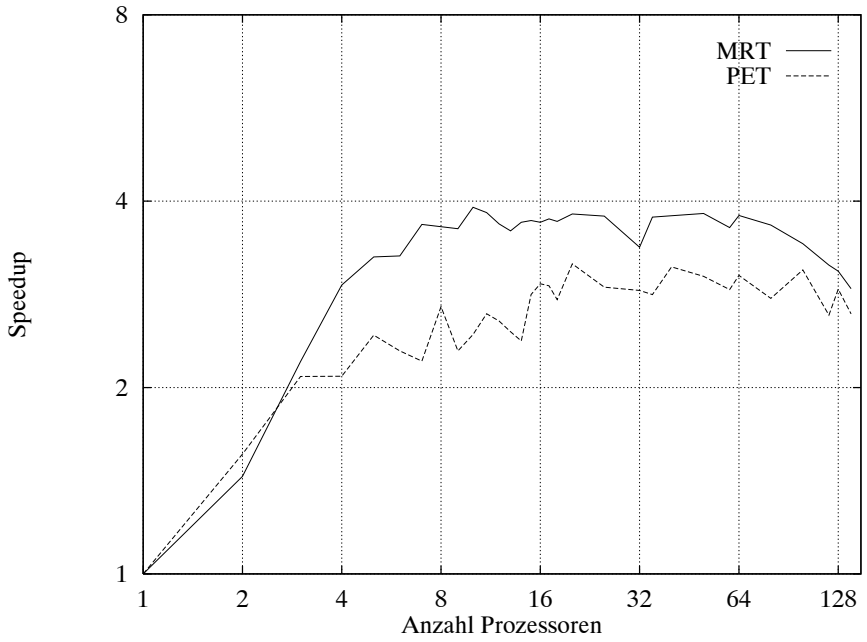


Abbildung 8.6: Speedup des Dateneinlesens

Dateien einlesen, die er aufgrund der Datenverteilung zu bearbeiten hat. Für das Einlesen wurden hier die Standard-C-Routinen zur Dateibehandlung verwendet[32], insbesondere wurden die Daten mit der Funktion *read* gelesen. Der erreichte Speedup bewegt sich bei den MRT-Daten bis in den Bereich um vier. Eine weitere Steigung wird nicht erreicht, da die Verwaltungskapazität für die gewählte Datenablage schnell erschöpft ist. Die Größe der PET-Schichtdaten ist gegenüber den MRT-Schichten kleiner, so daß dort der Effekt noch stärker auftritt. Die Schwankungen in den Kurven aus Abb. 8.6 deuten darauf hin, daß diese Messungen im Mehrbenutzerbetrieb stattfanden. Dies war, wie bereits erwähnt, auch der Grund für die Berücksichtigung der minimalen Zeit, die gemessen wurde. Eine Verbesserung ist hier durch eine andere Datenablage zu erreichen, die der Systemarchitektur besser angepaßt ist.

Schwellwertbestimmung

Die Schwellwertbildung kann innerhalb der einzelnen Schichten stattfinden, ohne daß es einer Synchronisation bedarf. Die Schwellwertoperation brauchte für die parallele Version nicht weiter angepaßt zu werden, da diese Unterroutine bereits auf Schichtebene arbeitet. Hier bewährt sich die modulare Programmierung dieser Routine, die beliebig große Datensätze verarbeiten kann. Jeder Knoten übergibt der Routine den eingelesenen Datensatz in einem entsprechend dimensionierten Feld. In den Speedup-Kurven aus Abb. 8.7 zeigen sich deutliche Stufen, die auf die Datenverteilung zurückgehen. Ein Stufe entsteht, wenn die Zahl der zu bearbeitenden Schichten pro Knoten auf allen Knoten das nächst niedrigere Niveau erreicht hat. Dies ist besonders deutlich bei den 15 Schichten der PET zu sehen. Bei 14 Knoten muß einer

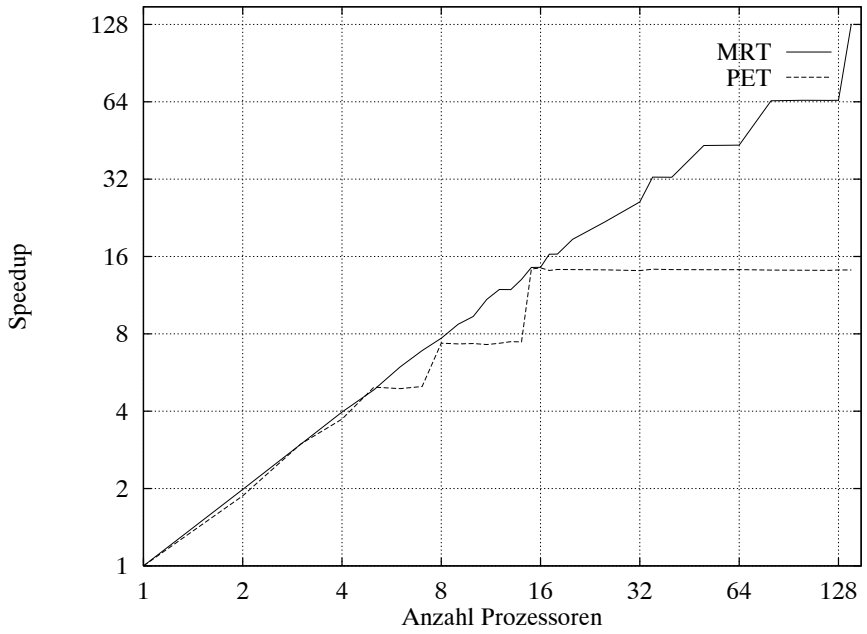


Abbildung 8.7: Speedup der Schwellwertoperation

der Prozessor 2 Schichten bearbeiten, während bei 15 Prozessoren nur jeweils eine PET-Schicht pro Knoten vorhanden ist. Da nur 15 PET-Schichten vorhanden sind, ist bei dieser Datenverteilung eine maximaler Speedup von 15 zu erreichen. Der tatsächlich erreichte Speedup bewegt sich bei 14,2 . Bei den MRT-Daten ergeben sich die gleichen Effekte, jedoch bleibt dieser Datensatz über einen weiteren Bereich skalierbar, da mehr (131) unabhängige Schichten vorhanden sind.

Eine weitere Leistungsteigerung bei den PET-Schichten ließe sich hier nur durch Aufbrechen der Schichtstruktur erreichen, die jedoch mit zunehmender Prozessorzahl einen immer geringer werdenden Einfluß auf die Gesamtlaufzeit hat.

Konturgenerierung

Ähnliche Effekte wie bei der Schwellwertbildung zeigen sich auch bei der Konturgenerierung, die ebenfalls in den einzelnen Schichten durchgeführt wird. Auch diese Routine kommt ohne Kommunikation aus und muß deshalb für die Parallelisierung nicht geändert werden. Die Routine ist bereits so aufgebaut, daß sie von der Zahl der Schichten unabhängig ist. Somit kann auch sie direkt angesprochen werden.

Das Speedup-Verhalten in Abb. 8.8 ist dem der Schwellwertoperation sehr ähnlich, wobei jedoch nicht ganz so hohe Speedup-Werte erreicht werden. Dies ist begründet in der ungleichen Lastverteilung innerhalb der Schichten. Aufgrund der Implementierung ist die benötigte Zeit für Schichten, die eine kleine zu konturierende Region enthalten, länger als für Schichten, die eine große zentrale Region besitzen. Aus diesem Grund ist die Zeit für die Bearbeitung einer Schicht nicht gleichverteilt, son-

dern variiert. Dies führt zu einer Verringerung des Speedup, da auf den langsamsten Knoten gewartet werden muß.

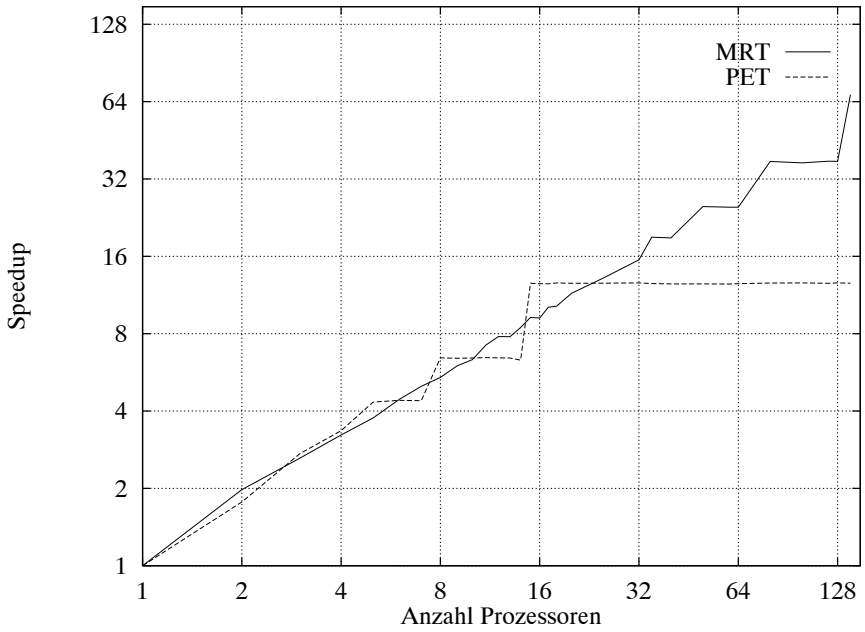


Abbildung 8.8: Speedup der Konturgenerierung

Punkteerzeugung

Wie bereits erwähnt hat die Punkteerzeugung nur einen sehr geringen Anteil an der Gesamtzeit, sie soll aber trotzdem untersucht werden. Die Punkteerzeugung kann ebenfalls innerhalb der lokalen Schichten auf den Prozessoren erfolgen. Der Algorithmus wurde gegenüber der sequentiellen Version etwas abgeändert, um für jede Prozessorzahl die gleichen numerischen Ergebnisse zu erreichen. Hierzu wird zunächst die globale Länge über alle Konturen bestimmt und allen Knoten bekanntgemacht. In jeder Schicht wird der erste Konturpunkt einer Konturliste als Punkt übernommen und anschließend im gleichen Abstand weitere erzeugt. Bei der sequentiellen Version geschah dies gleichmäßig über den ganzen Datensatz. Da bei der Bestimmung der Punkte die Lage der einzelnen Schicht im globalen System wichtig ist, erhält dieser Teilalgorithmus Informationen über die Datenzuteilung und berechnet hiermit die Positionen der Punkte im Gesamtkoordinatensystem. Bei der PET-Punkteerzeugung ist die Auflösungskompensation eingeschlossen, während die MRT-Punkteerzeugung eine globale Kommunikation enthält, um die MRT-Kontrollpunkte zu sammeln.

Abb. 8.9 zeigt die bei dieser Operation erreichten Speedup-Werte, die nicht besonders hoch liegen. Hierfür gibt es eine Reihe von Gründen. Zunächst ist eine globale Kommunikation unter allen Knoten notwendig, um die globale Konturlänge zu bestimmen. Diese Operation ist sehr zeitaufwendig, da alle Prozesse synchronisiert

werden müssen. Nach der Synchronisation findet Kommunikation statt, die mit zunehmender Knotenzahl immer aufwendiger wird. Der Speedup für die Erzeugung der MRT-Punkte fällt hierbei zusätzlich noch weiter ab, da gegen Ende der Operation die Punkte global gesammelt werden. Ein weiterer ungünstiger Effekt, der hier allerding nicht so schwerwiegend in Erscheinung tritt, ist die ungleiche Zahl von Punkten, die in den einzelnen Schichten erzeugt wird. Nichtzuletzt tritt auch hier das Problem auf, daß nur innerhalb der Schichten gearbeitet werden kann, was den Speedup auf die Zahl der PET-Schichten begrenzt.

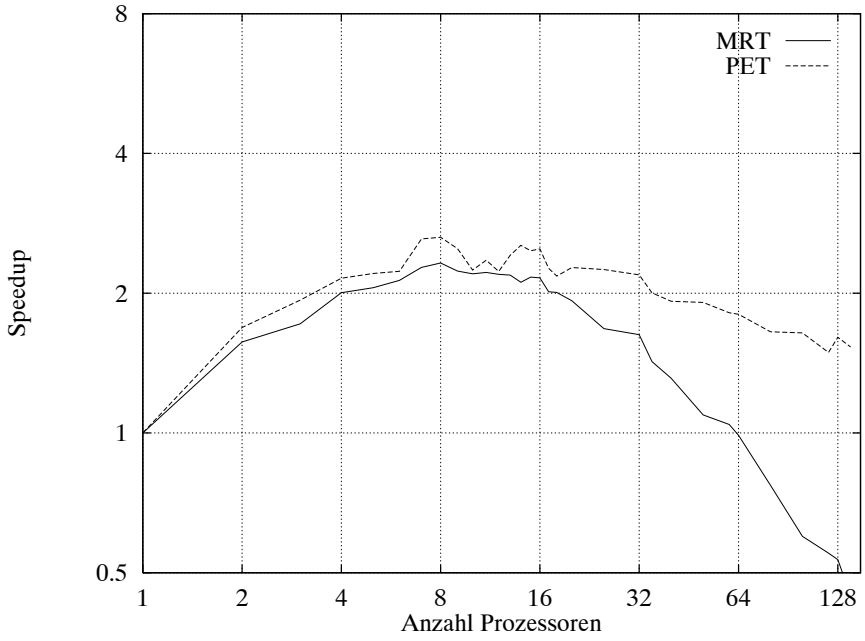


Abbildung 8.9: Speedup der Punkteerzeugung

Zur Verbesserung dieses Teilalgorithmus wäre es möglich die Anzahl der Punkte pro Schicht lokal festzulegen, um den globalen Austausch der Längen zu unterbinden. Dann bekommt man jedoch verschiedene numerische Ergebnisse für verschiedene Knotenzahlen, da die Konturen verschiedene Längen haben und so die Punkte teilweise dichter verteilt werden. Das Einsammeln der MRT-Punkte könnte mit asynchronen Kommunikationsroutinen zu einem Zeitpunkt geschehen, wenn die Knoten anderweitig beschäftigt sind, da diese Daten erst am Ende des Gesamtalgorithmus gebraucht werden. Diese Überlegungen sind jedoch erst sinnvoll, wenn zu noch höheren Prozessorzahlen übergegangen wird, so daß sich der Zeitanteil dieser Routine weiter bemerkbar macht.

Tiefenkartenerzeugung

Die Tiefenkartenerzeugung läuft innerhalb der einzelnen MRT-Schichten ab, bei der alle sechs Karten für jede Schicht bestimmt werden. Anschließend werden über eine globale Kommunikation allen Prozessoren die kompletten Tiefenkarten bekannt

gegeben. Für die globale Einordnung der lokalen Kontur wird der Routine bekanntgegeben, aus welchen Schichten die Konturlisten stammen.

Das Einordnen der Konturen geht vergleichsweise schnell. Selbst ein einzelner Knoten benötigt für die Tiefenkartenerstellung nur etwa 0,5s. Diese Zeit ist reine Rechenzeit, da bei einem Knoten keine Kommunikation stattfindet. Ordnen mehrere Knoten gleichzeitig die lokal vorhandenen Konturen ein, so ist eine weitere Verringerung der Rechenzeit zu erwarten. Auf der anderen Seite setzt mit zunehmender Zahl der Knoten eine immer komplexere globale Kommunikation ein, wobei die Tiefenkarten hier sehr umfangreich sind (1,5Mbyte). Aus diesem Grund ergibt sich der in Abb. 8.10 gezeigte Speedup-Verlauf, der mit zunehmender Prozessorzahl immer schlechter wird.

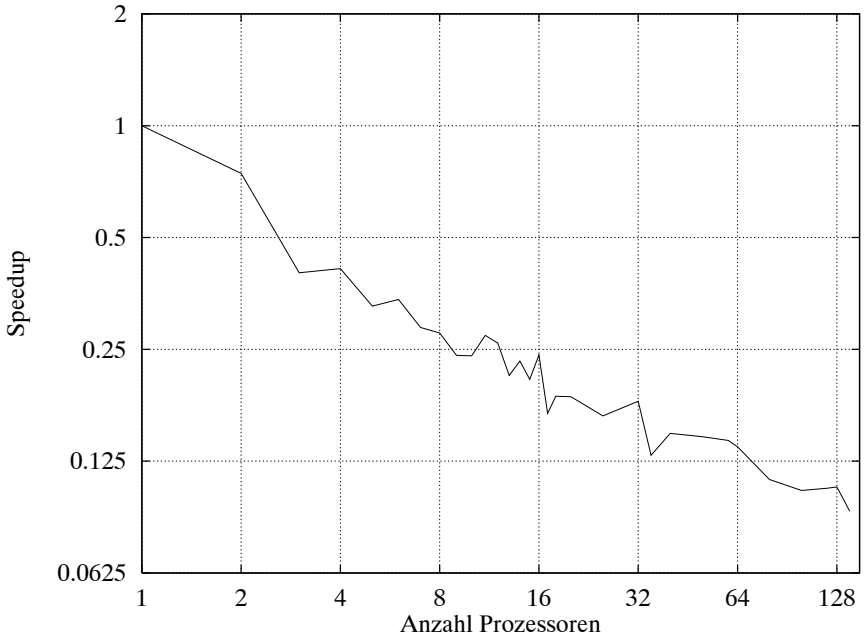


Abbildung 8.10: Speedup der Tiefenkartenerzeugung

Transformationsfindung

Die Bestimmung der Parameter geschieht über die Minimierung der Abstände der PET-Punkte zur MRT-Oberfläche, vgl. Abschnitt 6.3.3 . Die Verteilung der PET-Punkte ist, eng an die Schichtverteilung gekoppelt. Innerhalb eines Knotens sind nur diejenigen Punkte bekannt, die in den vom Prozessor zu bearbeitenden Schichten liegen. Eine Information über die zu bearbeitenden Schichten ist hier nicht nötig, da die PET-Punkte bereits auf ein globales System bezogen sind. Die weiterhin notwendigen MRT-Tiefenkarten sind lokal auf allen Knoten verfügbar. In dieser Routine tritt das Problem der Synchronisation der Einzelprozesse auf den Knoten auf, da die Anpassung für den Gesamtdatensatz als Ganzes durchgeführt werden soll. Hieraus folgt, daß ein Knoten ausgezeichnet werden muß, der die erhaltenen

Werte global sammelt und neue Suchrichtungen und -punkte zentral vorgibt. In diesem Fall übernimmt Knoten 0 diese Aufgabe. Alle anderen Knoten gehen in einen Wartezustand, indem sie auf Kommandos des Knoten 0 warten. Durch einen Broadcast der gerade aktuellen Parameter berechnen alle Knoten die für ihre PET-Punkte geltenden Abstände. Diese Werte werden über eine globale Kommunikation in Knoten 0 gesammelt. Nach Beenden der Optimierung schickt Knoten 0 allen Knoten die endgültigen Parameter, nach denen die lokalen PET-Punkte ein letztes Mal transformiert werden und abschließend alle Punkte auf Knoten 0 gesammelt werden. Dieser Knoten erhält diese Daten, um sie später an den Client zu senden.

Abb. 8.11 zeigt das Ergebnis der Speedup-Messung. Die Messung zeigt ein der PET-Konturgenerierung sehr ähnliches Ergebnis. Bis zur Zahl der PET-Schichten steigt der Speedup-Verlauf deutlich an, wobei auch hier eine Stufenstruktur erkennbar ist. Wie bereits bei der PET-Schwellwertbildung und der PET-Konturgenerierung läßt sich dieser Effekt auf die gewählte Datenverteilung zurückführen. Es fällt hierbei auf, daß die Stufen nicht so klar erkennbar sind, wie bei den vorgenannten Operationen. Dies hat im wesentlichen zwei Gründe. Auf der einen Seite existiert eine globale Kommunikation, die vor allem für den Abfall der Kurve oberhalb von 16 Prozessoren verantwortlich ist. Mit zunehmender Prozessorzahl wird der Aufwand für eine globale Kommunikation aufwendiger. Der zweite Grund ist die teilweise ungleichmäßig verteilte Anzahl der PET-Punkte innerhalb einer Schicht; dies zeigt sich auch deutlich in dem erreichten Speedup. So sind in der untersten PET-Schicht aufgrund der längeren Kontur wesentlich mehr Punkte vorhanden als in der obersten Schicht.

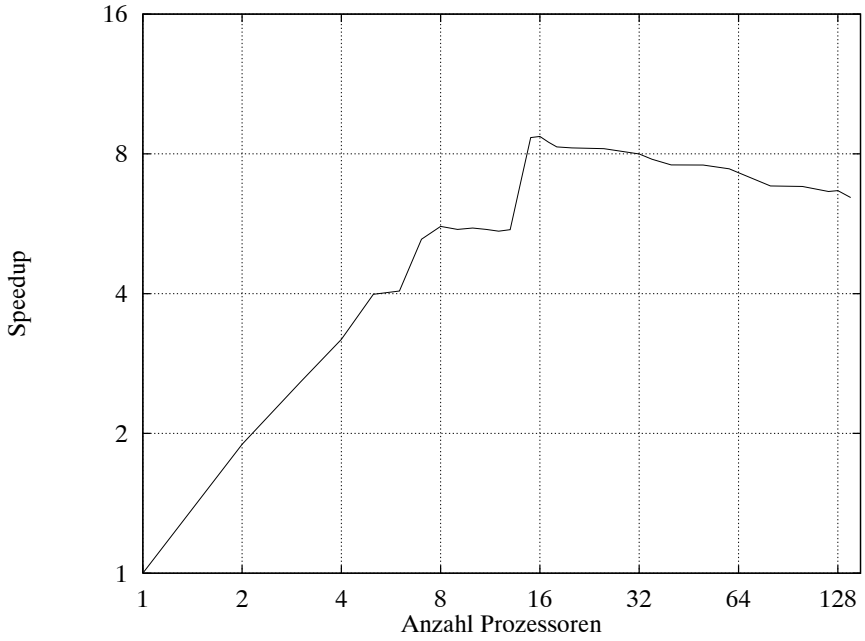


Abbildung 8.11: Speedup der Transformationsfindung

Eine Verbesserung dieser Situation ließe sich durch eine zusätzlich implementierte

globale Operation erreichen, die die Punkte gleichmäßiger auf die Prozessoren verteilt. Somit würde eine bessere Lastverteilung erreicht werden.

Datenneuanordnung

Die Datenneuanordnung läßt sich in zwei Teile untergliedern. Zunächst werden die neu errechneten PET-Schichten erzeugt. Dies kann innerhalb der Knoten, die PET-Schichten enthalten, parallel geschehen. Allerdings begrenzt diese Einschränkung den Speedup für diese Teiloperation auf die Zahl der vorhandenen PET-Schichten. Die neuberechneten Schichten werden anschließend mittels einer globalen Operation in Knoten 0 gesammelt. Die zweite Teiloperation ist die Ermittlung der MRT-Daten. Dazu wird in jedem Knoten ein Datenbereich angelegt, der den gesamten vorher neu berechneten PET-Schichten entspricht. Jeder Knoten trägt anschließend seine lokalen MRT-Daten an vorher berechneten Stellen ein. Nach dieser Operation findet eine globale Kommunikation statt, um die Gesamtschichten aus den Teilschichten zusammenzusetzten und dem Knoten 0 zu übergeben.

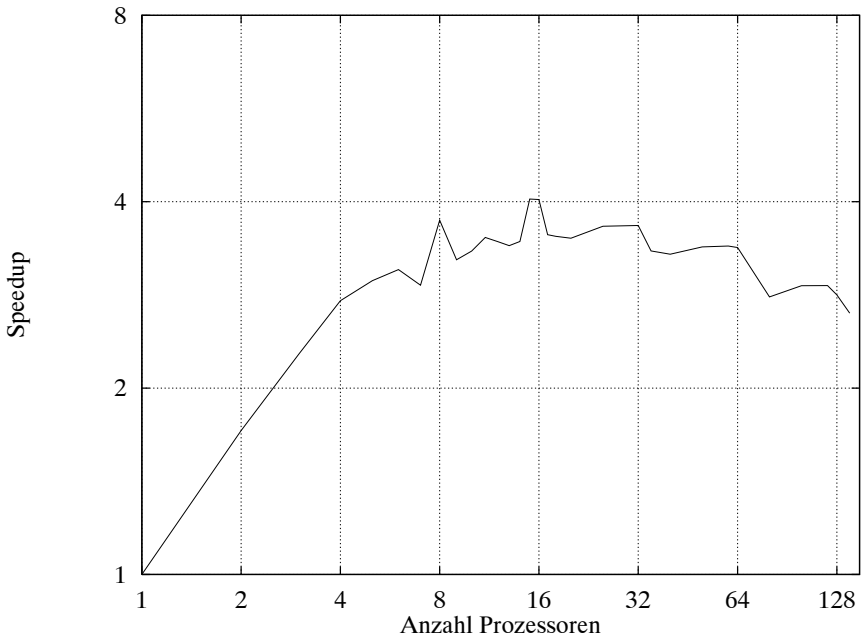


Abbildung 8.12: Speedup der Datenneuanordnung

Abb. 8.12 zeigt die für diese Routine erreichten Speedup-Werte, wobei das Maximum bei 15 Prozessoren mit einem Speedup von 4 erreicht wird. Danach fällt die Kurve leicht ab, da die Kommunikation eine immer stärkere Rolle spielt, die jedoch nicht so groß ist wie beispielsweise bei der MRT-Tiefenkartenerzeugung. Eine Verbesserung könnte hier dadurch erreicht werden, daß Teile der Kommunikation überlappend mit der Berechnung stattfinden. So könnten die Ergebnisse der ersten Operation gesammelt werden, während bereits mit der Berechnung der MRT-Teilschichten begonnen wurde.

8.2.3 Bewertung

Mit der hier vorgestellten parallelen Implementierung ist eine deutliche Reduzierung der Bearbeitungszeit verbunden. Der maximale Speedup liegt nahe bei der Anzahl der vorhandenen PET-Schichten. Dieses Verhalten hängt, wie gezeigt, von der gewählten Datenverteilung ab. Es stellt sich die Frage, inwieweit es sinnvoll ist, für den betrachteten Fall mehr als 8 Prozessoren zu verwenden. Ab dieser Prozessorenzahl ist keine deutliche Verbesserung des Zeitverhaltens und des damit verbundenen Speedup zu erkennen. Wie bereits gezeigt, hat die Client/Server-Kommunikation und das Einlesen der Daten einen erheblichen Zeitanteil, der für die Anwendung ebenfalls von Bedeutung ist.

Die AVS-Funktionalität, daß nur solche Daten neu zu berechnen sind, die sich aufgrund eines neu formulierten Client-Auftrages ändern, wurde bisher nicht weiter in Betracht gezogen. Da es häufig vorkommt, daß mehrere Aktionen an einem Datensatzpaar stattfinden, fällt bei Verwendung dieser Funktionalität das Einlesen der Daten weg, da die Daten bereits auf den Knoten vorhanden sind. Das Hauptaugenmerk richtet sich dann auf den eigentlichen Anpassungsalgorithmus. Je nach Auftrag müssen nur einzelne Programmabschnitte durchlaufen werden, da sich für andere Abschnitte die Eingangsdaten und -parameter nicht ändern und das Ergebnis schon vorliegt. Muß beispielsweise nur die Transformationsfindung neu gestartet werden, fällt die sehr teure Erstellung der MRT-Tiefenkarten weg. Die Antwortzeit sinkt für diese Operation von $30s$ bei einem Knoten auf 6 s bei 16 Knoten. Vergleicht man dies mit der Zeit von $28s$ bei 16 Knoten, die die Bearbeitung eines völlig neuen Auftrages benötigt, zeigen sich hier nochmal deutlich die Vorteile dieser modular implementierten AVS-Funktionalität.

Die verwendete Kommunikation wirkt sich negativ auf die erreichbaren Speedups aus. Zum Zeitpunkt der Messungen waren die verwendeten globalen Kommunikationsroutinen noch nicht endgültig installiert, sondern stützten sich auf andere Kommunikationsroutinen. Mit Verwendung besser implementierter Kommunikationsroutinen ist eine weitere Verbesserung der erreichten Speedups zu erwarten. Für eine darüberhinaus gehende Erhöhung des Speedup und eine bessere Skalierbarkeit muß die verwendete Schichtstruktur aufgebrochen werden. Hierfür muß eine Reihe von neuen, anders arbeitenden Algorithmen entwickelt werden, die nicht von der Schichtstruktur abhängen. Eine einfache Erhöhung des Speedup und der damit verbundenen besseren Skalierbarkeit bei der Transformationsfindung läßt sich durch eine gleichmäßige Verteilung der Punkte auf die vorhandenen Knoten erreichen. Dies wird insbesondere sinnvoll, wenn man weitere, aufwendigere Optimierungsstrategien implementieren möchte, die den Zeitanteil der Transformationsfindung steigen lassen.

Eine andere wichtige Frage ist, ob es sinnvoll ist, einen Parallelrechner interaktiv zu bedienen. Man erhält zwar für den Benutzer eine deutliche Senkung der Antwortzeit, die Rechenleistung des Parallelrechners wird jedoch nur zeitweise genutzt. Es ist also keine gleichmäßige Auslastung des Rechners möglich. Zur Zeit wird nur ein

Prozeß pro Knoten gestartet. Findet eine Bewertung der Ergebnisse durch den Benutzer am Schirm statt, liegt der Rechner brach, bis ein neuer Auftrag eintrifft. In diesem Zusammenhang wäre es sehr sinnvoll, andersartige Prozesse auf den Knoten zuzulassen, die dann arbeiten, wenn der Server-Prozeß auf neue Anfragen wartet.

Kapitel 9

Zusammenfassung

In dieser Arbeit wurde ein Verfahren zur Verknüpfung der funktionalen Information der Positronen-Emissions-Tomographie (PET) mit der aus der Magnet-Resonanz-Tomographie (MRT) gewonnenen anatomischen Information untersucht. Zu diesem Zweck wurden den PET-Schichtbildern räumlich entsprechende MRT-Schichtbilder überlagert, um so ein Zuordnung der in den Daten enthaltenen Informationen zu ermöglichen.

Aus der Reihe der vorgeschlagenen Anpassungsalgorithmen wurde ein Verfahren implementiert, das sich auf das von Pelizzari et al. beschriebene Kopf-und-Hut-Verfahren stützt. Bei diesem implementierten Algorithmus werden in zwei Vorverarbeitungsschritten die Datensätze auf die für die Anpassung nötigen Informationen reduziert. Aus den PET-Daten wird eine Reihe von Punkten erzeugt, die die Außenkontur dieser Daten beschreiben. Die Außenkontur der MRT-Daten wird in sechs Tiefenkarten abgelegt, die einen schnellen Zugriff auf die Lage der Oberflächenpunkte ermöglicht. Mit Hilfe dieser beiden Außenkonturbeschreibungen findet eine geometrische Anpassung der beiden Konturdaten statt. Bei der Anpassung werden die Parameter einer affinen Transformation ermittelt, die eine Neuanordnung der MRT-Daten in Relation zu den PET-Daten ermöglicht. Die hierbei erreichte Genauigkeit liegt im Bereich von 1,85 mm durchschnittlicher Abweichung der PET-Punkte von der MRT-Oberfläche. Hier kann nur der Fehler einer Oberflächenanpassung betrachtet werden. In weiteren Schritten ist zu klären, ob sich die genannte Abweichung auch auf die inneren Strukturen übertragen läßt.

Das Verfahren wurde mit Hilfe des Visualisierungs- und Entwicklungssystems AVS implementiert. Innerhalb dieser Umgebung sind die notwendigen Teiloperationen des Gesamtalgorithmus als AVS-Module formuliert und implementiert worden. Somit lassen sie sich in die vorhandene Umgebung direkt integrieren und mit Standard-Bedienungselementen des AVS beeinflussen. Auf diese Weise entsteht eine Benutzeroberfläche, die der Anwender intuitiv bedienen kann. Darüber hinaus bietet AVS die Möglichkeit, interaktiv in das Verarbeitungsnetz aus erstellten Modulen einzugreifen. Auf diese Weise sind einfache Erweiterungen des Netzes möglich, indem weitere eigene oder AVS-Module hinzugefügt werden. Diese Erweiterungen betreffen bei-

spielsweise zusätzliche Darstellungen der Daten oder Umformungen in gewünschte Dateiformate. Da die implementierten Teilalgorithmen als Module vorliegen, können zwischen die Teiloperationen weitere Module eingefügt werden, die das erstellte Verfahren an grundsätzliche Änderungen der Eingangsdaten anpassen.

Der Algorithmus hat sich in der Anwendung als robust und flexibel erwiesen. Da für die einzelnen Datensätze eine Reihe von beschreibenden Parametern angegeben wird, ist eine sehr flexible Anpassung von Daten unterschiedlichster Größe und Auflösung möglich. Es hat sich herausgestellt, daß die interaktive Änderung von Parametern, die den Gesamtalgorithmus betreffen, sehr nützlich ist. So ist beispielsweise die Bestimmung des Schwellwertes für die vorhandenen Binärbilderzeugungen entscheidend für die nachfolgende Anpassung. Oftmals ist es allerdings nur durch interaktiven Eingriff möglich, diesen Schwellwert zu bestimmen, da die vorhanden Daten in den unterschiedlichsten Formen vorliegen und somit keine automatische Bestimmung dieser Schwelle möglich ist. Die für die gesamte Berechnung der hier verwendeten beiden Datensätze benötigte Rechenzeit bewegt sich im Bereich von 5 *min* auf dem System Stardent GS2002, wobei anders dimensionierte Datensätze eine deutliche Verlängerung oder Verkürzung der Bearbeitungszeit bedeuten können. Die AVS-Eigenschaft der Module, nur veränderte Daten neu zu bearbeiten, stellt einen erheblichen Vorteil dar, wenn nur Teile der Verarbeitungskette geändert wurden. In diesem Fall müssen nur die davon beeinflußten Module aktiv werden, was zum Teil einen erheblichen Zeitvorteil bedeutet.

Entscheidendes Kriterium für die Bestimmung der Transformationsparameter ist die Minimierung des Abstandes der Oberflächen aus den verschiedenen Datensätzen. Bei dieser Minimierung tritt eine Reihe von Schwierigkeiten auf, die neben der Mehrdeutigkeit der Anpassung durch vorhandene Symmetrien in den Daten auch das Problem lokal vorhandener Minima betrifft. Hierfür ist eine Reihe von Einstellparametern implementiert worden, die eine flexible Handhabung der Optimierung ermöglichen. Die Frage der automatischen Optimierung konnte in diesem Zusammenhang nicht erschöpfend geklärt werden, da hierzu noch eine Reihe von Untersuchungen nötig ist, die in Rahmen dieser Arbeit nicht möglich war. Es ist vorstellbar, für die Optimierung Modelle zu verwenden, die bereits bekannte Information über die Anpassung nutzen und nicht so allgemein formuliert sind wie die hier verwendeten Algorithmen. Es bietet sich beispielsweise an, zunächst nur eine translatorische Anpassung vorzunehmen, an die sich dann die Ermittlung der Rotationsparameter der affinen Transformation anschließt.

Es hat sich gezeigt, daß ein erhebliches Maß an Rechenleistung für die Anpassung nötig ist. Die damit verbundene lange Antwortzeit ist für einen direkten klinischen Einsatz ungünstig. Aus diesem Grund wurde untersucht, ob sich der im ersten Teil implementierte Ansatz auch auf einem massiv-parallelen System implementieren läßt und welche Vorteile sich daraus ergeben.

Zu diesem Zweck wurde zunächst ein Client/Server-Modell entwickelt, um die dem Anwender bereits bekannte Schnittstelle zum System zu erhalten. Die Anwendung wurde in zwei Teile zerlegt. Der Server-Prozeß läuft hierbei auf einem beliebigen

UNIX-System und enthält die gleiche Funktionalität der bereits erwähnten AVS-Module. Insbesondere findet auf dem Server die Anpassung der Daten statt, die anschließend über ein Netz an den Client-Prozeß übermittelt werden. Der Client stellt ein Bearbeitungsnetzwerk dar, das nur noch die Bedienungselemente und Anzeigemöglichkeiten des ursprünglichen Verfahrens beinhaltet. Mit Hilfe der Bedienungselemente formuliert der Client einen Auftrag, der vom Server bearbeitet wird. Die ermittelten Ausgangsdaten des Server werden mit den Anzeigemodulen des Client dargestellt. Client und Server besitzen eine definierte Schnittstelle. Solange sich Client und Server an diese Definition halten, spielt die konkrete Implementierung für die Funktionalität eine untergeordnete Rolle. Somit ist es auch möglich, für den Server andere Rechnerstrukturen, wie beispielsweise das in dieser Arbeit verwendete System Intel Paragon, zu nutzen. Der Erfolg des Client/Server-Modells hängt sehr stark von der die Systeme verbindenden Netzwerkverbindung ab, da eine langsame Datenübertragung die Vorteile einer andersartigen Rechnerarchitektur zunichte machen kann. In dieser Arbeit verbindet das Client/Server-Modell die Vorteile der Visualisierung und Bedienung durch AVS mit der Nutzung schneller massiv-paralleler Rechnerarchitekturen.

Neben einer sequentiellen Version des Server auf zwei Workstation-Systemen wurde ein parallele Version für das massiv-parallele System Intel Paragon erstellt. Das System besteht aus einer Reihe von Rechenknoten, die mit Hilfe des Konzepts Message Passing programmiert wurden. Für die Parallelisierung wurde die vorhandene Datenparallelität ausgenutzt. Die in jedem Datensatz vorhandenen Schichten wurden gleichmäßig auf die Knoten verteilt und innerhalb der Schichten parallel verarbeitet. Die Untersuchung der Teilalgorithmen des Server ergaben sehr unterschiedliche Werte für die einzelnen Beschleunigungswerte. Hierbei spielt besonders die benötigte Kommunikation eines Teilalgorithmus eine wesentliche Rolle. Es konnte eine Verringerung der Antwortzeit um den Faktor 4 erreicht werden. Entscheidend ist, daß diese Werte bereits bei 8 bis 10 Prozessoren erreicht werden und mit zunehmender Prozessorzahl nicht weiter zu steigern sind. Hierfür gibt es eine Reihe von Gründen. So ist die Zeit, die der Server für die sequentielle Datenübertragung zum Client benötigt, konstant, während das Einlesen der Schichtdaten maximal um den Faktor 4 beschleunigt werden konnte. Die Aufteilung der Schichten auf die Knoten hat einen weiteren wichtigen Einfluß, da nur ganze Schichten an die Knoten vergeben werden können. Der Beschleunigungsfaktor ist somit auf die Anzahl der an einer Teiloperation beteiligten Schichten begrenzt. In dem hier gezeigten Fall waren 15 PET-Schichten vorhanden. In diesem Zusammenhang ist die Frage interessant, ob es sich lohnt, die Schichtstruktur weiter aufzuspalten, um eine weitere Beschleunigung zu erhalten. Ein anderes zu untersuchendes Thema ist, wie die hier vorgestellte interaktive Nutzung des Parallelrechners sich auf die Auslastung desselben auswirkt. Für den hier betrachteten Anwenderkreis ist eine minimale Antwortzeit wichtig, wobei zwischen den einzelnen Aufträgen einige Zeit vergeht, in der der Rechner ungenutzt ist. Hierbei stellt sich die Frage, wie man diese ungenutzte Zeit für andere Aufgaben nutzen kann.

Literaturverzeichnis

[1] Advanced Visual Systems Inc. : AVS Developer's Guide. 4. Auflage, 1992

[2] Advanced Visual Systems Inc. : AVS Module Reference. 5. Auflage, 1993

[3] Advanced Visual Systems Inc. : AVS User's Guide. 4. Auflage, 1992

[4] G.S. Almasi, A. Gottlieb: Highly Parallel Computing. Benjamin/Cummings, Redwood City, 1989

[5] N.M. Alpert, J.F. Bradshaw, D. Kennedy et al.: The Principal Axis Transformation - A Method for Image Registration. Journal Nuclear Medicine 31:1717-1722, 1990

[6] J.R. Anderson, S.C. Stropher, X. Xu et al.: Error Bound for Five Registration Techniques Based on High Resolution MRI. Annals of Nuclear Medicine, Scientific Paper 30, Conference Brain PET '93 in Akita (Japan), 1993

[7] H. Bässman, P.W. Besslich: Konturorientierte Verfahren in der digitalen Bildverarbeitung. Springer, Heidelberg, 1989

[8] H.-G. Buchholz: Entwicklung eines Programms zur automatisierten räumlichen Zuordnung von PET- und MRT-Bildern. Diplomarbeit FH Aachen Abt. Jülich, 1993

[9] R. Esser, R. Knecht: Intel Paragon XP/S — Architecture and Software Environment. In: H.-W. Meuer: Supercomputer '93:121-141, Springer, Heidelberg, 1993

[10] A.C. Evans, C. Beil, S. Marrett et al.: Anatomical-Functional Correlation Using an Adjustable MRI-Based Region of Interest Atlas with Positron Emission Tomography. Journal Cerebral Blood Flow and Metabolism 8:513-530, 1988

[11] A. Evans, S.Marrett, L. Collins et al.: Anatomical-Functional Correlative Analysis of the Human Brain Using Three Dimensional Imaging Systems. SPIE 1092:264-274, 1989

[12] R. Fletcher: Practical Methods of Optimization. Wiley-Interscience, New York, 1987

[13] M.J. Flynn: Very High-Speed Computing Systems. Proceedings of IEEE 54(12): 1901-1909, 1966

[14] F. Friedrichs: Graphenalgorithmen auf massiv-parallelen Rechnern. Erscheint als: Berichte des Forschungszentrums Jülich, Jülich, 1993

[15] A. Gamboa-Aldeco, G.T.Y. Chen: Correlation of 3D surfaces from multiple modalities in medical imaging. SPIE proceedings 460-466, 1986

[16] P.E. Gill, W. Murray, M.H. Wright: Practical Optimization. Academic Press, London, 1981

[17] P.A. Gloor: Synchronisation in verteilten Systemen. B.G. Teubner, Stuttgart, 1989

[18] R.C. Gonzalez, P.Wintz: Digital Image Processing. Addison-Wesley, Reading, 1987

[19] L. Gottesfeld Brown: A Survey of Image Registration Techniques. ACM Computing Surveys 24(4):325-376, 1992

[20] R. Grzeszczuk, K.K. Kim, D.N. Levin et al.: Retrospective Fusion of Radiographic and MR Data for Localisation of Subdural Electrodes. Journal of Computer Assisted Tomography 16(5):764-773, 1992

[21] H. Handels: Automatische Analyse mehrdimensionaler Bilddaten zur Diagnoseunterstützung in der MR-Tomographie. Dissertation RWTH Aachen, Shaker, 1992

[22] R.W. Hockney, C.R. Jesshope: Parallel Computers 2. Adam Hilger, Bristol, 1988

[23] B.L. Holman, R.E. Zimmerman, G.T.Y. Chen et al.: Computer-Assisted Superimposition of Magnetic Resonance and High-Resolution Technetium-99m-HMPAO and Thallium-201 SPECT Images of the Brain. Journal of Nuclear Medicine 8:1478-1484, 1991

[24] Y. Huang et al.: 3D Matching of Structural (CT,MR) and Functional (PET) Brain Image Data. Computer Assisted Radiology, CAR '91, Springer, Heidelberg, 1991

[25] K. Hwang, F.A. Briggs: Computer Architecture and Parallel Processing. McGraw-Hill, New York, 1984

[26] Intel Supercomputer Systems Division: Paragon OSF/1 C Compiler User's Guide. Intel, Beaverton, 1993

[27] Intel Supercomputer Systems Division: Paragon OSF/1 C System Calls – Reference Manual. Intel, Beaverton, 1993

[28] Intel Supercomputer Systems Division: Paragon OSF/1 User's Guide. Intel, Beaverton, 1993

[29] H. G. Jacob: Rechnergestützte Optimierung statischer und dynamischer Systeme. Springer, Heidelberg, 1992

[30] J. Kapouleas, A. Alavin, W. Alves et al.: Registration of Three-Dimensional MR and PET Images of the Human Brain without Markers. Radiology 181(3):731-739, 1991

[31] K.J. Kearfott, D.A. Rottenberg, R.J.R. Knowles: A new Headholder for PET, CT and NMR Imaging. Journal Computer Assisted Tomography 9:1217-1220, 1984

[32] B.W. Kernighan, D.M. Ritchie: The C Programming Language. Prentice Hall, 1977

[33] E. Krestel(Hrsg.), E. Ammann(Mitarb.): Bildgebende Systeme für die medizinische Diagnostik. Siemens. Berlin, 1988

[34] D.N. Levin, C.A. Pelizzari, G.T.Y. Chen et al.: Retrospective Geometric Correlation of MR, CT, and PET Images. Radiology 169:817-823, 1988

[35] D. Meyer-Ebrecht: Vorlesung Digitale Bildverarbeitung I & II. Lehrstuhl für Meßtechnik, RWTH Aachen, 1992

[36] H.W. Müller-Gärtner, J.M. Links, J.L. Price et al.: Measurement of Radiotracer Concentration in Brain Gray Matter Using Positron Emission Tomography: MRI-Based Correction for Partial Volume Effects. Journal Cerebral Blood Flow Metabolism 12:571-583, 1992

[37] Open Software Foundation: OSF/1 Operating System, User's Guide. Prentice Hall, 1992

[38] S. Ortmanns: Modelle der Kooperation von Graphik-Workstations und Höchstleistungsrechnern bei Visualisierungsproblemen. Berichte des Forschungszentrums Jülich, Jül-2719, Jülich, 1993

[39] M. Papageorgiou: Optimierung, Statische, Dynamische, Stochastische Verfahren für die Anwendung. Oldenbourg, München, 1991

[40] C.A. Pelizzari, G.T.Y. Chen, D.R. Spelbring et al.: Accurate Three-Dimensional Registration of CT, PET, and/or MR Images of the Brain. Journal Computer Assisted Tomography 13:20-26, 1989

[41] U. Pietrzyk, K. Herholz, W.-D. Heiss: Three-dimensional Alignment of Functional and Morphological Tomograms. Journal of Computer Assisted Tomography 14(1):51-59, 1990

[42] M.J.D. Powell: An efficient method for finding the minimum of a function of several variables without calculating derivatives. Computer Journal 7:155-163, 1964

[43] W.K. Pratt: Digital Image Processing. Wiley-Interscience, New York, 1991

[44] Sun Microsystems, Inc.: Network Programming Guide. 1990

[45] A.S. Tanenbaum: Computer Networks. Prentice-Hall, Englewood Cliffs, New Jersey, 2. Auflage, 1989

[46] M. Santifaller: TCP/IP und NFS in Theorie und Praxis: UNIX in lokalen Netzen. Addison-Wesley, Reading, 1990

[47] L.R. Schad, R. Boesecke, W. Schlegel et al.: Three Dimensional Images Correlation of CT, MR, and PET Studies in Radiotherapy Treatment Planning of Brain Tumors. Journal Computer Assisted Tomography 11(6):949-954

[48] L.A. Shepp, Y. Vardi: Maximum Likelihood Reconstruction for Emission Tomography. IEEE Transactions on Medical Imaging MI-1(2):113-122, 1982

[49] F.A. Stichnoth (Hrsg.): Handbuch der Magnet-Resonanz-Tomographie. Blackwell-Wiss.-Verl., Berlin, 1992

[50] K.Wienhard, R. Wagner, W.-D. Heiss : PET, Grundlagen und Anwendugen der Positronen-Emissions-Tomographie. Springer, Heidelberg, 1989

Danksagung

Die vorliegende Arbeit wurde als Diplomarbeit in Elektrotechnik am Lehrstuhl für Technische Informatik und Computerwissenschaften der Rheinisch-Westfälischen Technischen Hochschule (RWTH) Aachen erstellt.

Dem Lehrstuhlinhaber Herrn Prof. Dr. F. Hoßfeld danke ich für die Möglichkeit, die Arbeit am Zentralinstitut für Angewandte Mathematik (ZAM) des Forschungszentrums Jülich (KFA) anfertigen zu können.

Mein besonderer Dank gilt Herrn Dr. P. Weidner für die Betreuung der Arbeit und konstruktive Diskussionen. Frau Dr. E. Rota Kops und Herrn L. Tellmann vom Institut für Medizin des Forschungszentrums Jülich danke ich für die zahlreichen Anregungen und Diskussionen, die mir einen tiefen Einblick in die medizinische Sichtweise des in der Arbeit behandelten Problems gaben.

Weiterhin möchte ich die gute und kameradschaftliche Atmosphäre unter den Studenten und Doktoranden am ZAM hervorheben, die zu einem guten Gelingen dieser Arbeit beigetragen hat. An dieser Stelle möchte ich auch allen meinen Freunden und meinen Eltern danken, die mich auf dem Weg durch mein Studium begleitet und unterstützt haben, dessen Abschluß diese Arbeit darstellt.