Fachhochschule Aachen
Campus Jülich

Fachbereich: Medizintechnik und Technomathematik
Studiengang: Scientific Programming
Matrikelnummer: 869601

FH AACHEN
UNIVERSITY OF APPLIED SCIENCES

Vereinigung der Richtungsinformationen von Nervenfaserverläufen aus PLI-Bildern zur Überbrückung verschiedener Skalen

Bachelorarbeit von

Andreas Müller

17. August 2015

Eigenständigkeitserklärung

Diese Arbeit ist von mir selbstständig angefertigt und verfasst. Es sind keine anderen als die angegebenen Quellen und Hilfsmittel benutzt worden.

Ort und Datum

Andreas Müller

Diese Arbeit wurde betreut von:

Erstprüfer: Prof. Dr. Johannes Grotendorst (JSC)
Zweitprüfer: Dr. Markus Axer (INM-1)

Diese Arbeit wurde erstellt im Forschungszentrum Jülich am Jülich Supercomputing Centre (JSC) in Kooperation mit dem Institut für Neurowissenschaften und Medizin (INM-1)

Abstract

Um den Verlauf von Nervenfasern im menschlichen Gehirn verfolgen zu können, wurde am Institut für Neurowissenschaften und Medizin des Forschungszentrum Jülich die Methode *Polarized Light Imaging* (PLI) entwickelt. Dabei wird ein Gehirn post mortem in 60 µm dünne Scheiben (ca. 2000 pro Gehirn) geschnitten und durch ein Mikroskop aufgenommen.

Mit einer Einzelaufnahme des gesamten Schnittes könnte die zur Faserdarstellung benötigte Auflösung nicht erreicht werden. Daher werden die Schnitte jeweils kachelweise aufgenommen. Die so erzeugten Daten werden vom PLI-Rekonstruktionsworkflow in eine 3D-Karte zusammengefasst. Hierbei werden unter anderem die Faserausrichtungen der Projektionen auf die Schnittebene und die Winkel der Fasern zur Schnittebene bestimmt. In weiteren Schritten werden eine Maske zur Trennung von Hirn und Hintergrund berechnet und die Kacheln zu einem Gesamtschnitt zusammengefasst. Danach werden die einzelnen Schnitte registriert, d.h. strukturell passend übereinandergelegt, sodass eine Faserverfolgung durch mehrere Schnitte möglich ist.

Diese 3D-Karte, das Volumen, konnte bisher nicht verifiziert werden. Ziel der Arbeit ist die Entwicklung einer Methode zur richtungserhaltenden Auflösungsreduktion. Es werden mehrere Methoden zum Zusammenfassen von Richtungsinformationen von Nervenfasern entwickelt, die bei Reduktion der Auflösung eine korrekte Darstellung der Richtungen von Nervenfasern ermöglichen sollen. Dadurch können in Zukunft PLI-Bilder beispielsweise mit Aufnahmen eines MRT verglichen werden.

Darüber hinaus wurde das Programm parallelisiert, um den Zeitaufwand der Berechnungen signifikant zu senken.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Struktogrammverzeichnis

1 Einleitung

Eines der anspruchsvollsten und komplexesten Vorhaben unserer Zeit ist der Versuch, die Funktionsweise des menschlichen Gehirns zu verstehen und dessen Struktur zu simulieren. Um dieses Ziel zu erreichen, unterstützt die europäische Union das *Human Brain Project* als eines von zwei langzeitgeförderten Flagship-Projekten [BMBF]. Unter dessen Dach arbeiten Wissenschaftler aus vielen verschiedenen Ländern und Bereichen, wie z.B. Neurowissenschaften, Informatik, Physik und Mathematik, zusammen. Ziel ist es, zukünftig das Gehirn in seiner Gesamtheit simulieren zu können.

Auch das Forschungszentrum Jülich ist mit verschiedenen Instituten beteiligt. Dazu zählen das Institut für Neurowissenschaften und Medizin (INM) und das Jülich Supercomputing Centre (JSC).

Um das menschliche Gehirn simulieren zu können, muss zunächst herausgefunden werden, wie die etwa 86 Mrd. Nervenzellen [AC+09] miteinander verbunden sind. Schon bei dieser Frage steht man vor einer großen Herausforderung, da alle modernen bildgebenden Verfahren, wie die Magnetresonanztomografie (MRT), in ihrem Auflösungsvermögen begrenzt sind. So hat z.B. die kleinste Struktur, die durch ein klinisches MRT dargestellt werden kann, eine Größe von etwa 2 mm. Post mortem sind Auflösungen von einigen 100 µm möglich.

Nervenfasern haben Durchmesser zwischen 0,5 µm und 10 µm [SHL11], weshalb eine individuelle Faserverfolgung mit MRT-Aufnahmen auf Grund der Dichte von Fasern im menschlichen Gehirn nicht möglich ist.

Um Faserdarstellung zu ermöglichen, wurde am INM-1 eine Methode der Bildgebung entwickelt, die in diese Bereiche der Auflösungskraft vordringt: Polarized Light Imaging (PLI).

Vorbereitend dafür wird ein Gehirn post mortem in 60 µm dünne Scheiben geschnitten und mit polarisiertem Licht beleuchtet. Der Anteil des beim Durchscheinen des Schnittes doppelt gebrochenen Lichtes wird aufgenommen und mit Hilfe des PLI-Rekonstruktionsworkflows der Verlauf der Faser in einem Bildpunkt bestimmt. Die dabei erreichte kleinste darstellbare Strukturgröße liegt bei 1,3 µm, knapp über der Dicke der dünnsten Fasern, was grundsätzlich zur Faserdarstellung ausreicht.

Um die durch PLI gewonnenen Richtungsinformationen von Fasern in einem Pixel verifizieren zu können, müssen sie mit bekannten Informationen verglichen werden. Um die Daten beispielsweise mit MRT-Aufnahmen vergleichen zu können, muss die Auflösung der Bilder so weit reduziert werden, dass diese mit der Auflösung eines MRT-Bildes übereinstimmen.

Da Nervenfasern Orientierungen anstelle von Richtungen besitzen, sind die Informationen bei der Reduktion der Auflösung nur schwierig zusammenzufassen. Verschiedene mögliche Ansätze hierfür werden in dieser Arbeit vorgestellt.

2 PLI und Rekonstruktionsworkflow

Um aus einzelnen Schnitten des Gehirnes ein räumliches Fasermodell zu erzeugen, müssen die Richtungsinformationen der Fasern bestimmt werden. Da diese nicht direkt aufgenommen werden können, müssen sie nach Aufnahme der Schnitte in einem anschließenden Workflow bestimmt werden.

2.1 Polarized Light Imaging

Da die dünnsten Fasern einen Durchmesser von etwa 0,5 µm haben, muss ein bildgebendes Verfahren etwa diesen Auflösungbereich erreichen, um auch dicht liegende Fasern darstellen zu können. Dafür macht sich das Verfahren des Polarized Light Imaging eine wesentliche Eigenschaft der Nervenfasern zu Nutze: Doppelbrechung des Lichtes durch Myelinisierung der Nervenfasern.

Die meisten Nervenfasern im Gehirn sind von einer dünnen Schicht aus Myelin umgeben (Abb. 2.1) [AK14], die unter anderem der elektrischen Isolation dient. PLI basiert auf der doppelbrechenden Eigenschaft des Myelins mit optischer Achse in Faserrichtung. Zur Bestimmung der Faserrichtung muss also die Richtung der optischen Achse der Myelinscheide bestimmt werden.

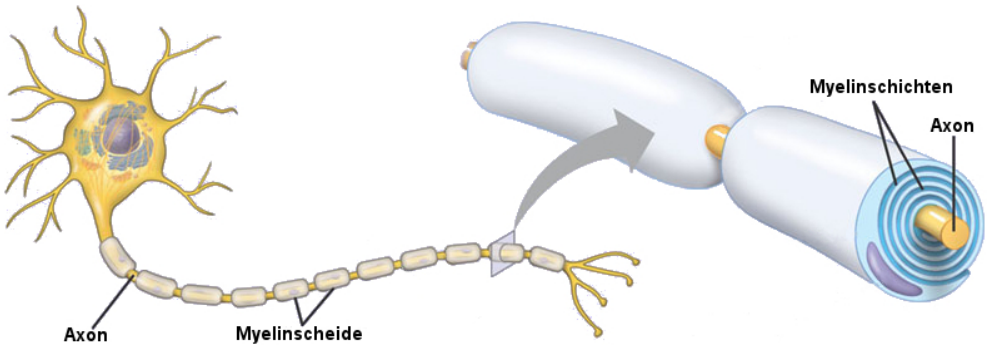


Abbildung 2.1: Aufbau einer Nervenfaser

Der Gehirnschnitt wird in dem dafür entwickelten Versuchsaufbau mit zirkular polarisiertem Licht einer bestimmten Wellenlänge λ durchleuchtet. An den myelinisierten Fasern des Schnittes erfährt es eine Doppelbrechung und elliptische Polarisation. Um die Polarisation zu vermessen, wird ein weiterer Polarisationsfilter eingesetzt, dessen Transmissionsachsen senkrecht zum ersten stehen (Abb. 2.2).

Weil die eingesetzte Kamera eine begrenzte Auflösung hat, müssen die Schnitte mit Hilfe eines Mikroskopes aufgenommen werden. Dieses Mikroskop erhöht die Auflösungskraft des Verfahrens, hat allerdings den Nachteil, dass der Schnitt nicht in seiner Gesamtheit aufgenommen werden kann. Deshalb muss der Gesamtschnitt in Teilkacheln unterteilt werden. Um bei der Aufnahme der 2048×2048 Pixel großen Kacheln keine Informationen zu verlieren, werden diese überlappend aufgenommen. Informationen über den Verlauf der Brechungsachse können nicht aus einem einzelnen Bild extrahiert werden. Dafür sind Informationen über die Stärke der Doppelbrechung aus unterschiedlichen Richtungen nötig. Deshalb werden pro Kachel 18 Bilder aufgenommen, jeweils mit um 10° vergrößertem Winkel ρ der Polarisationsfilter bzw. Verzögerungsplatte. Da der Winkel, in dem das Licht auf eine einzelne Faser trifft, sich abhängig von der Position des Polarisationsfilters ändert, ändert sich auch die Stärke der Doppelbrechung und somit die Intensität des aufgenommenen Lichtes. So ergibt sich für jedes Pixel über diese 18 Bilder gesehen ein individueller Helligkeitsverlauf. Dieser kann durch eine Sinuskurve beschrieben werden, aus der sich die räumliche Orientierung der Faser rekonstruieren lässt.

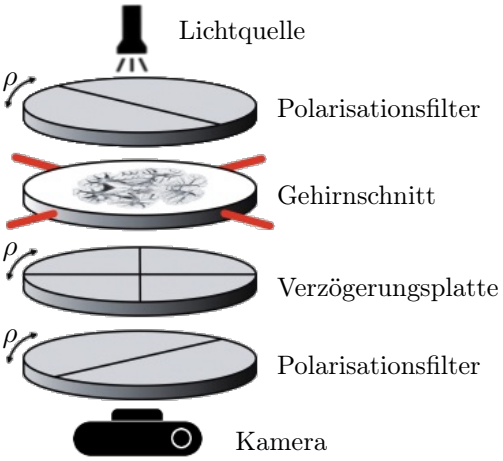


Abbildung 2.2: PLI Aufbau

2.2 PLI-Rekonstruktionsworkflow

Um aus einzelnen Kacheln mit je 18 Bildern ein 3D-Gesamtbild zu erzeugen, durchlaufen die Daten den PLI-Rekonstruktionsworkflow. Dieser berechnet in mehreren Schritten die räumliche Orientierung, fügt die Kacheln zu einem Schnitt zusammen und diese Schnitte zu einem Gesamtvolumen.

2.2.1 Direktion, (normierte) Transmittanz und Retardierung

In einem ersten Schritt wird aus dem Helligkeitsverlauf der 18 Pixel eine Sinuskurve approximiert. Der Verlauf dieser Sinuskurve lässt sich nun durch drei Werte beschreiben: Die Verschiebungen an x- und y-Achse sowie die Amplitude der Kurve.

Die Verschiebung entlang der x-Achse wird **Direktion** genannt. Diese entspricht dem Winkel der Projektion der Faser innerhalb der Schnittebene zur vorher festgelegten Null-Grad-Richtung.

Die Verschiebung der Sinuskurve in y-Richtung, d.h. die Höhe der Ruhelage der Kurve ist die Hälfte der **Transmittanz**. Sie ist ein Maß für die Lichtdurchlässigkeit des Schnittes an dieser Stelle. Um sie für weitere Berechnungen zu vereinheitlichen und eventuelle Unregelmäßigkeiten, die in der Belichtungszeit oder der Kamera selbst begründet liegen, zu entfernen, wird die **Transmittanz normiert**. Dazu dividiert man

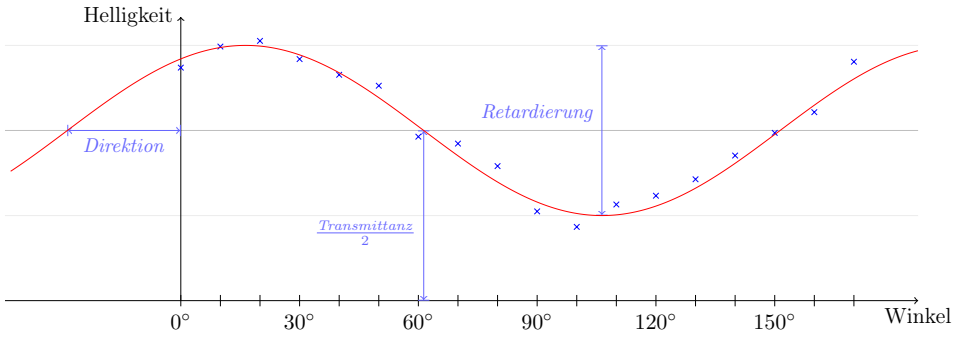
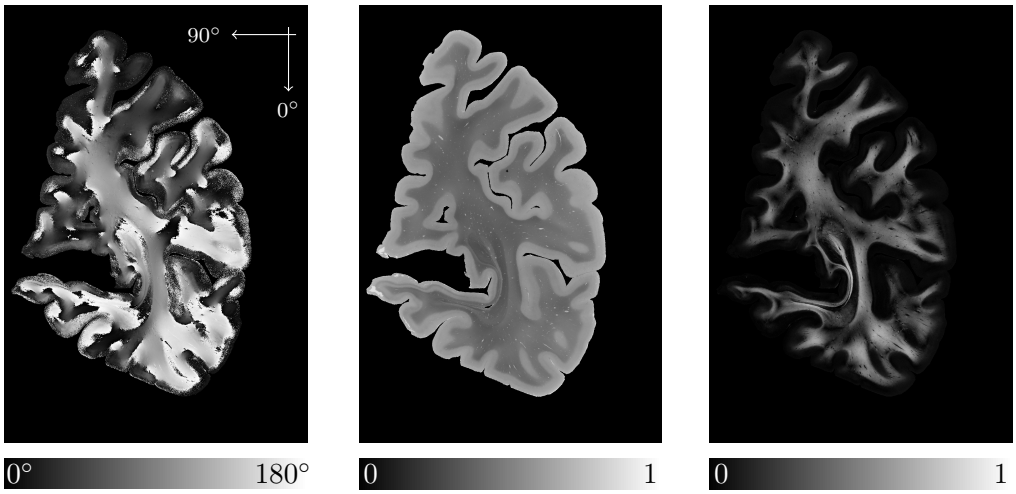


Abbildung 2.3: Bestimmung von Direktion, Transmittanz und Retardierung aus dem Helligkeitsverlauf eines Pixels der 18 Bilder einer Kachel

die Werte der Transmittanz pixelweise durch die Transmittanzwerte eines Leerbildes, also einer Aufnahme ohne Gehirnschnitt.

Die Amplitude der Sinuskurve, die Differenz zwischen Maximum und Minimum, ist der **Retardierungswert**. Er ist ein Maß für die Stärke der Myelinisierung in diesem Punkt, da die Amplitude der Sinuskurve dann groß wird, wenn die Doppelbrechung größer ist oder die Ausrichtung der Myelinscheide in der Schnittebene liegt.



(a) Direktion (b) normierte Transmittanz (c) Retardierung

Abbildung 2.4: Beispielbilder für Direktion, normierte Transmittanz und Retardierung. Die 0-Grad-Richtung im Direktionsbild verläuft senkrecht von oben nach unten. Alle Winkel werden im Uhrzeigersinn gemessen. Der für diese Arbeit verwendete Datensatz wurde aus der linken Hemisphere des Hirns eines Patienten mit Multisystematrophie gewonnen. Die Schnittebene ist frontal.

Für jedes Bild werden pro Pixel nicht die 18 Werte, sondern die Modalitäten (Direktion, Transmittanz und Retardierung) gespeichert. Dieser Vorgang ist nahezu verlustfrei, da man davon ausgehen kann, dass die Abweichungen zur Sinuskurve durch Aufnahmefehler und -ungenauigkeiten begründet werden können. Die Speicherung der Charakteristika der Sinuskurve erzeugt erste Eigenschaften der Nervenfasern, hat aber auch den positiven Nebeneffekt, dass durch sie die zu verarbeitenden Daten auf ein Sechstel ihrer Größe reduziert werden können. So sinkt beispielsweise der Speicherplatzbedarf eines Schnittes, der in 50×30 überlappende Kacheln aufgeteilt wurde, von 50×30 (Kacheln) $\cdot\, 2048 \times 2048$ (Pixel/Bild) $\cdot\, 18$ (Bilder/Kachel) $\cdot\, 4$ (Bytes/Pixel) $= 421{,}875\,\text{GiB}$ auf knapp über 70 GiB.

2.2.2 Stitching

Unter Zuhilfenahme der normierten Transmittanz werden die einzelnen Kacheln wieder zu einem Gesamtschnitt zusammengefügt, was man Stitching nennt. Dabei werden in benachbarten Kacheln markante Punkte gesucht, die dann beim Stitching durch Verschiebung der Kacheln zueinander übereinandergelegt werden (Abb. 2.5). Dadurch wird es möglich, einzelne Schnitte im Gesamten zu verarbeiten. Ebenso würde eine kachelweise Verarbeitung der Daten nicht garantieren können, dass jeder reale Punkt des Gehirnes nur in einem Pixel enthalten ist.

Ebenfalls wird so die Datenmenge um knapp die Hälfte reduziert, da die einzelnen Kacheln in x- und y-Richtung zu etwa 30% überlappen, wie bereits in Kapitel 2.1 erwähnt.

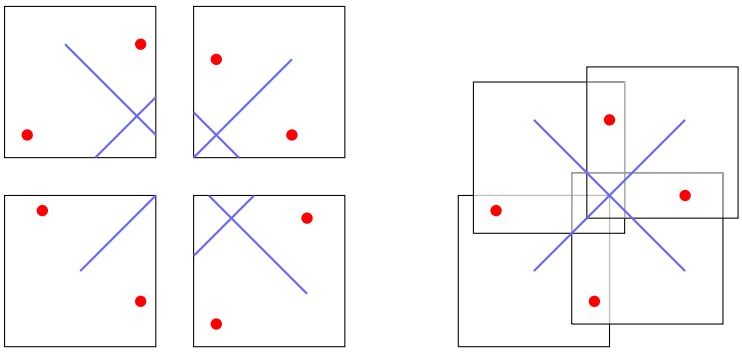


Abbildung 2.5: Schematische Darstellung des Stitchings

2.2.3 Segmentierung

Da das Transmittanzbild die klarste Abgrenzung zwischen Gehirn und Hintergrund, aber auch zwischen grauer und weißer Substanz besitzt, wird auf dieser Basis die Segmentierung durchgeführt. Dabei wird mit einem Region-Growing-Algorithmus eine Maske berechnet, die die Bildpunkte bestimmten Gehirnregionen zuordnet (Abb. 2.6). Dazu gehören Hintergrund, graue Substanz und weiße Substanz. Die weiße Substanz ist dabei der Bereich des Hirns, in dem hauptsächlich stark myelinisierte Nervenfasern verlaufen und weniger Nervenzellen liegen. Umgekehrt liegen die Nervenfasern in der grauen Substanz nicht so dicht beeinander.

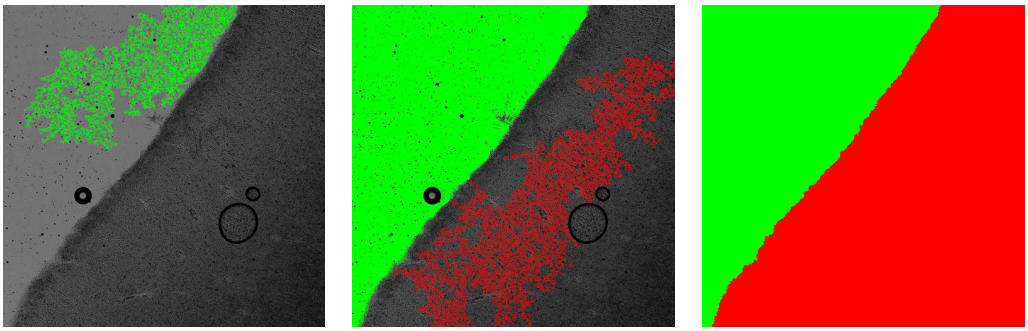


Abbildung 2.6: Beispielverlauf des Region-Growings

Durch den Einsatz der Maske kann die Menge der zu verarbeitenden Daten weiter reduziert werden, da etwa ein Sechstel aller Daten zum Hintergrund gehört und bei den weiteren Berechnungen nicht mehr beachtet werden muss. Dafür benötigt man etwa 1% mehr Speicherplatz, da man die Daten der Maske im kleinsten Datentyp (1 Byte pro Pixel) abspeichern kann, weil man nur drei Werte unterscheiden muss. Diese sind die Indices für Hintergrund, graue Substanz und weiße Substanz.

2.2.4 Inklination

Nach dem Stitching der einzelnen Kacheln kann der Inklinationswinkel α zwischen Faser und Schnittebene berechnet werden (Abb. 2.7). Dabei wird unter anderem jedem Retardierungswert ein Inklinationswinkel zugeordnet.

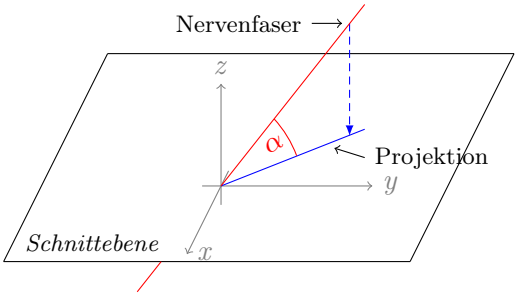


Abbildung 2.7: Inklinationswinkel

Für diese Berechnung muss jedoch auf dem Histogramm über alle Daten des Gehirnes der Wert gefunden werden, dem ein Winkel von 0° zugeordnet wird.

Weil die Formel zur Berechnung der Inklination einen Gewichtungsfaktor abhängig von der Transmittanz enthält, das Transmittanzbild jedoch stark verrauscht ist, muss dieses Bild vorher geglättet werden. Dafür wird ein Nachbarschaftsmedianfilter angewendet. Das bedeutet, dass jedes Pixel so verändert wird, dass dessen Wert dem Median aller ungefilterten Werte in einem runden Bereich um das ursprüngliche Pixel entspricht.

2.2.5 Registrierung der einzelnen Schnitte

In einem letzten Schritt werden die einzelnen Schnitte zu einem 3D-Bild, dem PLI-Volumenbild, zusammengefügt. Dafür reicht es nicht, sie so gut wie möglich übereinander zu legen. Das liegt an der Verformung der einzelnen Schnitte beim Schneidevorgang.

Nach dem Schneiden ist es nicht möglich, beim Zusammenlegen wieder die ursprüngliche Form zu erhalten.

Es wäre gut zu wissen, an welcher absoluten Position sich markante Punkte des Schnittes befunden haben. Aus diesem Grund wird vor dem Erzeugen eines Schnittes ein Bild des noch nicht geschnittenen Gehirnes aufgenommen, das sogenannte Blockface-Bild. Mit dessen Hilfe ist es möglich zu wissen, wo ein Schnitt im Gesamtbild platziert werden muss, um die ursprüngliche Struktur zu erhalten.

Eine weitere Hürde sind die unterschiedlichen Eigenschaften von grauer und weißer Substanz. Unter anderem wegen ihrer hohen Myelindichte verformt sich weiße Substanz beim Schneiden anders als graue, weshalb das Bild unter Zuhilfenahme von Streckung, Scherung und anderen Transformationen modifiziert werden muss, bis der transformierte Schnitt in markanten Punkten mit dem Blockface-Bild übereinstimmt.

Darüber hinaus müssen auch die einzelnen Schnitte aufeinander registiert werden, da das Blockfacebild ein reines Lichtbild ist und keine feinen Strukturen darstellen kann.

3 PLI-Bilder

Fasst man alle in Kapitel 2 beschriebenen Daten zusammen, so erhält man für ein Gehirn etwa 2000 Schnitte mit je etwa 1500 Kacheln. Für jede dieser Kacheln werden 18 Bilder aufgenommen. Das ergibt insgesamt bereits 54 Millionen Dateien für die Rohdaten des Gehirnes. Hinzu kommen die Daten für Retardierung, Transmittanz, Direktion, Inklination und Segmentierung. Abgesehen von der Inklination gibt es einen Datensatz jeweils für ungestitchte und gestitchte Bilder. Dabei ist zum einen bereits die riesige Anzahl der Bilder ein Problem für manche Dateisysteme, zum anderen ist die Speicherung der gestitchten Bilder in vielen üblichen Datenformaten nicht möglich. Deshalb wurde HDF5 als Datenformat ausgewählt.

3.1 HDF5

Das zur Speicherung der Bilder gewählte Datenformat Hierarchical Data Format 5 (HDF5) wurde vom National Center for Supercomputing Applications entwickelt und ist die Weiterentwicklung des 1995 vorgestellten HDF4. Wesentliche Neuerungen in Version 5 waren die Aufhebung der unsigned int-Grenze für Dateigrößen, die Dateien auf 2 GiB limitiert hat, und der Support für parallelen I/O [HDF5].

Hauptanwendungsbereich ist die Speicherung großer Datenmengen in wissenschaftlichen Anwendungen. So ist z.B. die Verwendung des HDF Pflicht für Produktionsdaten beim NASA Earth Observing System [NASA].

Die Dateien selbst sind hierarchisch organisiert. Genau wie in einem Filesystem hat jedes Objekt innerhalb der Datei einen Pfad, unter dem es erreichbar ist.

Der Pfad in jeder HDF5-Datei beginnt mit dem Wurzelverzeichnis „/“. In diesem Verzeichnis und den darunterliegenden Pfadelementen hat jedes Objekt einen Index, über den dessen Name abgefragt werden kann. Der Name wird zum Öffnen benötigt.

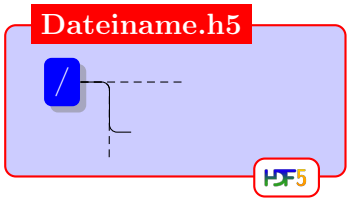


Abbildung 3.1: Wurzelverzeichnis

3.1.1 Gruppen

Die einfachste Art von Objekten sind die sogenannten Gruppen. Sie sind vergleichbar mit Ordnern auf Dateisystemebene und können beliebig viele weitere Objekte aufnehmen, darunter auch weitere Gruppen. Die Verschachtelungstiefe ist dabei nicht beschränkt.

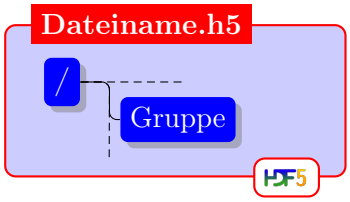


Abbildung 3.2: Gruppe

3.1.2 Datensätze

Nutzdaten, die in HDF5 abgespeichert werden sollen, werden üblicherweise in Datensätzen abgelegt. Diese haben eine beliebige Anzahl Dimensionen mit jeweils wählbarer, beliebiger Größe. Diese Eigenschaften des Datensatzes werden beim Erzeugen festgelegt und sind im Nachhinein nicht mehr änderbar. Wie jedes Objekt haben auch Datensätze einen Namen, über den man auf sie zugreifen kann.

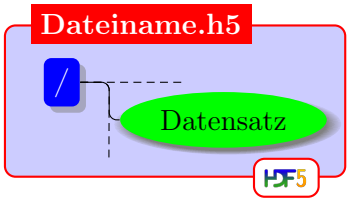


Abbildung 3.3: Datensatz

Um auch mit Datensätzen, die nicht vollständig in den Arbeitsspeicher passen, umgehen zu können, muss man diese stückweise verarbeiten. Aus diesem Grund kann man **Unterbereiche** von Datensätzen anwählen. Parameter sind dabei die Verschiebung des Startpunktes zum Einlesen in jeder Dimension (Offset), die Anzahl der einzulesenden Blöcke pro Dimension (Count), die Anzahl der Elemente pro Dimension, nach der ein weiterer Block eingelesen wird (Stride) und die Anzahl der Elemente der Dimensionen eines Blockes (Block) (Abb. 3.4).

Das Auswählen von Unterbereichen ermöglicht zudem parallelen I/O, in dem die Daten z.B. blockweise und nicht überlappend von den einzelnen Prozessoren eingelesen werden.

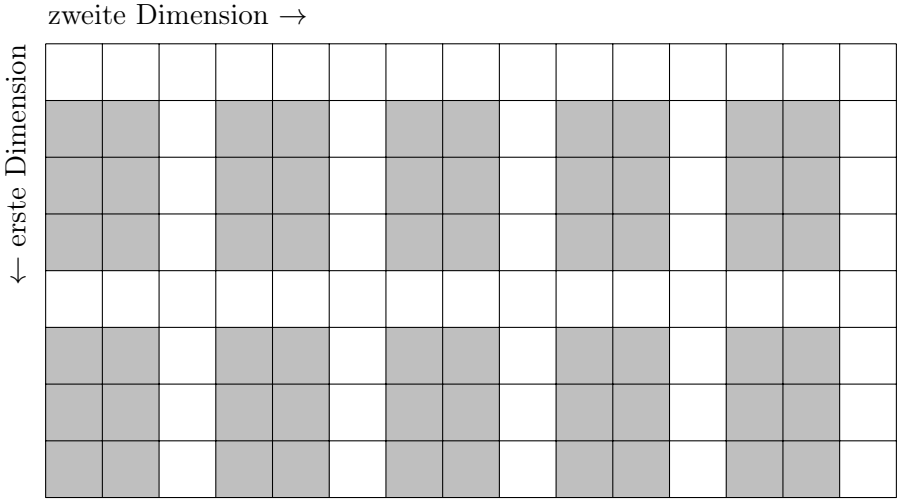


Abbildung 3.4: Die dunkel markierten Pixel eines Bildes werden ausgewählt, wenn man in einem Datensatz der Größe 8x15 einen **Unterbereich** auswählt mit folgenden Parametern:
Offset: (1,0) – Count: (2,5) – Stride: (4,3) – Block: (3,2)

3.1.3 Attribute

Zu jedem Objekt unterhalb des Wurzelverzeichnisses kann man Attribute hinzufügen, um Metadaten zu speichern. Die Daten selbst sind technisch gesehen wieder ein Datensatz. Sie bieten allerdings weniger Funktionen als normale Datensätze. So gibt es z.B. nicht die Möglichkeit Unterbereiche auszuwählen.

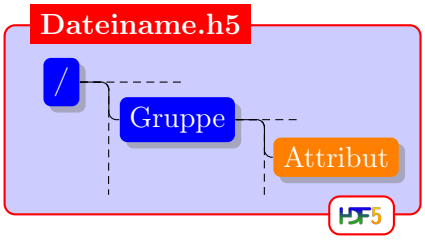


Abbildung 3.5: Attribut

3.1.4 Links

Bei Links ist zwischen internen und externen Links zu unterscheiden. Ein interner Link ermöglicht es, dass ein Datensatz oder eine Gruppe über mehrere Pfade innerhalb der Datei erreichbar ist, ohne die Daten doppelt speichern zu müssen. Bei der Verarbeitung der Objekte einer Datei muss keine Rücksicht auf eventuelle Links genommen werden. Öffnet man einen Link, so wird automatisch das verlinkte Objekt geöffnet.

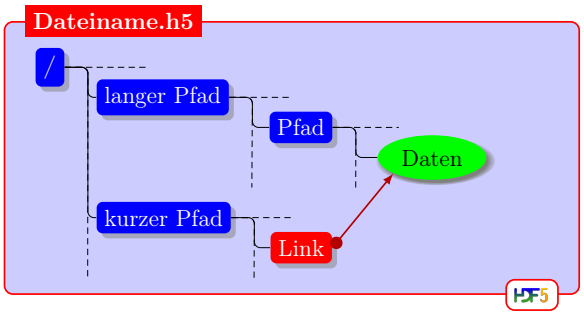


Abbildung 3.6: Beispiel für die Verwendung eines internen Links. Der Datensatz „`Daten`“ ist sowohl unter „`/langer Pfad/Pfad/Daten`“, als auch unter „`/kurzer Pfad/Daten`“ erreichbar.

Externe Links ermöglichen es, die vorhandenen Daten auf mehrere Dateien zu verteilen. Öffnet man einen externen Link, wird intern die Zieldatei geöffnet und dann das verlinkte Objekt. Auch dieser Vorgang geschieht implizit und bedarf keiner besonderen Beachtung seitens des Nutzers. Für diesen sieht es so aus, als befände sich das Zielobjekt innerhalb der geöffneten Datei.

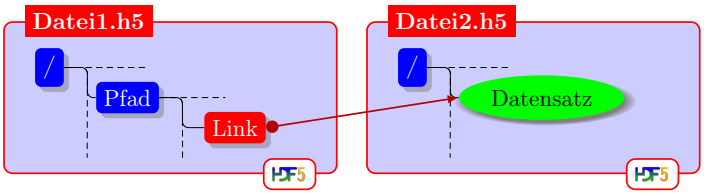


Abbildung 3.7: Beispiel für die Verwendung eines externen Links. Die in `Datei2.h5` liegenden Daten „`Datensatz`“ sind auch über den Pfad „`/Pfad/Datensatz`“ in der Datei `Datei1.h5` erreichbar.

3.2 Gewählte Struktur

Für den PLI-Workflow wird pro Schnitt eine eigene HDF5-Datei mit den einzelnen Datensätzen angelegt. Um diese Schnitte organisiert abzulegen, legt man für jedes Gehirn eine zentrale HDF5-Datei und einen Ordner mit dem Namen des Hirns an, z.B. „Brain“. Die HDF5-Datei „Brain.h5“ enthält die Links zu den HDF5-Dateien aller verfügbaren Schnitte, die im Ordner „Brain“ liegen.

Aufgrund der enormen Menge an Rohdaten übersteigt ein Schnitt die Marke von 100 GiB. Alle Rohdaten werden dabei als ein Bild nebeneinander abgelegt. Benötigt man nur einzelne Kacheln, so sind diese als Unterbereich auszulesen. Um eine saubere Trennung der Daten zu erhalten, werden die Daten ein weiteres Mal aufgeteilt. Die Datei, die bisher alle Daten eines Schnittes enthält, wird so modifiziert, dass sie nur noch Links enthält. Diese Links zeigen auf nach Art der Daten unterteilte HDF5-Dateien. Diese liegen in einem Unterordner, dessen Name Hirnnamen und Schnittnummer enthält, z.B. „Brain_125“. Für Rohdaten, Retardierung, Transmittanz, Segmentierung, Kalibrierung, Direktion und Inklination wird je eine Datei erzeugt. Die Transmittanzdatei enthält z.B. normale, normierte und gefilterte Transmittanz.

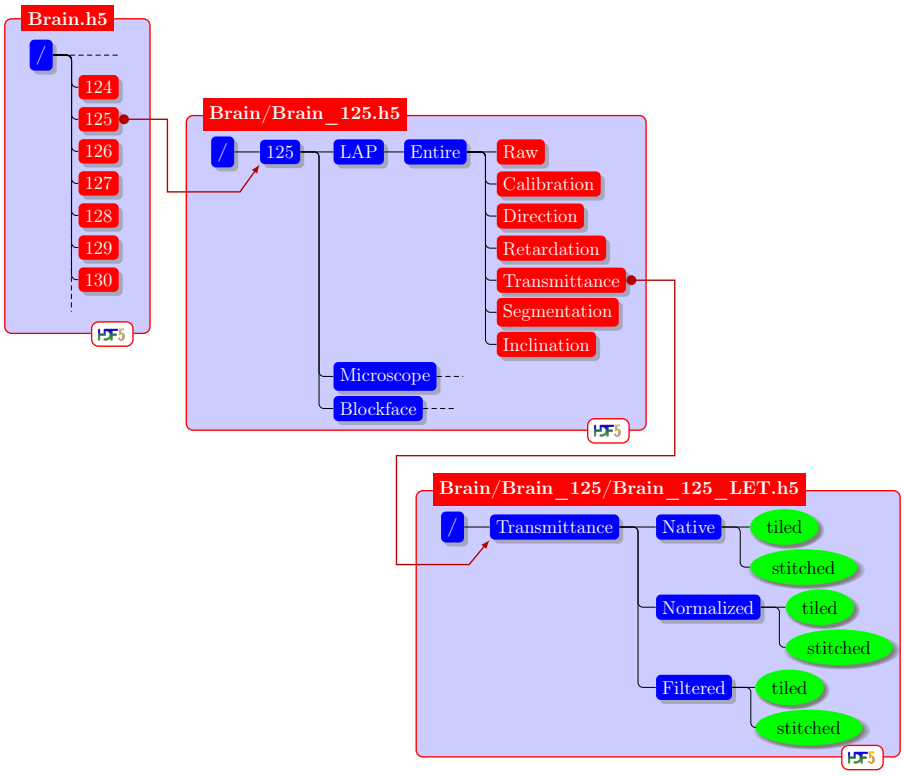


Abbildung 3.8: Ein Ausschnitt aus der gewählten HDF5-Struktur. Eine Datei pro Gehirn (z.B. „Brain.h5“) enthält Links zu allen Schnitten, die in einem Ordner mit Hirnnamen (z.B. „Brain“) liegen. Diese wiederum enthalten Links zu den verschiedenen Datensätzen des Schnittes, die in einem Unterordner mit Hirnnamen und Schnittnummer (z.B. „Brain_125“) liegen.

4 Korrektheit der Bilder

Die Berechnungen in Kapitel 2.2 basieren auf physikalischem und mathematischem Wissen, jedoch ist es bisher nicht möglich, die Korrektheit der Programmergebnisse zu verifizieren. Aufgrund von anatomischen und physiologischen Kenntnissen über Faserverläufe lassen sich nur die Tendenzen der Orientierung der größten Faserbahnen überprüfen. Abweichungen von wenigen Grad in den Winkeln einer Faser, die den Verlauf im 3D-Modell verfälschen, lassen sich nicht erkennen. Eine manuelle Prüfung der Bilder ist schwer realisierbar, da pro Gehirn bis zu 2000 Schnitte mit einer Auflösung im Gigapixel-Bereich überprüft werden müssen.

Es muss eine Methode entwickelt werden, die die Ergebnisse des PLI-Rekonstruktionsworkflows bestätigen kann.

4.1 Verifizierung durch Vergleich mit MRT-Bildern

Eine Möglichkeit ist der Vergleich der PLI-Volumenbilder mit Bildern, die mittels diffusionsgewichteter Magnetresonanztomographie (dMRT) erzeugt wurden. Die bei MRT-Aufnahmen erzielte Auflösung ist wesentlich geringer als bei PLI.

MRT-Bilder können als korrekt angesehen werden, da es ein etabliertes und gut untersuchtes Verfahren ist. Mindestens ebenso wichtig ist, dass eine MRT-Aufnahme **vor** dem Erzeugen der einzelnen Schnitte gemacht werden kann. Dadurch enthalten diese Bilder zum einen keine durch das Schneiden möglicherweise verfälschten Informationen, zum anderen beeinflusst das MRT als nicht-destruktives Verfahren die späteren PLI-Aufnahmen nicht.

4.2 Analyse der Daten

Die Richtungsinformationen der Nervenfasern liegen in sphärischen Koordinaten vor. Dies sind der Winkel der Faser in der Schnittebene (die Direktion δ), der Winkel zwischen der Faser und ihrer Projektion auf die Ebene (die Inklination α), und die Länge der Faser. Definitionsgemäß ist diese Länge immer 1, da mit Orientierungen gearbeitet wird, nicht mit Vektoren.

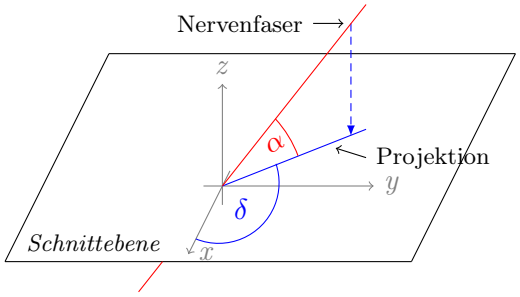


Abbildung 4.1: Lage der Winkel

Da die Arbeit mit den vorliegenden Kugelkoordinaten sehr rechenintensiv ist, da keine linearen Zusammenhänge bestehen, werden diese vor der Weiterverarbeitung in kartesische Koordinaten umgerechnet. Die allgemeinen Formeln dafür lauten:

$$\begin{aligned} x &= r \cdot \sin(\theta) \cdot \cos(\phi) \\ y &= r \cdot \sin(\theta) \cdot \sin(\phi) \\ z &= r \cdot \cos(\theta) \end{aligned} \tag{4.1}$$

Der Winkel ϕ entspricht der Definition der Direktion δ, diese kann für ϕ eingesetzt werden. Der Winkel θ ist der Winkel zwischen der z-Achse und der Nervenfaser. Da wir α als Winkel zwischen Faser und Projektion auf die xy-Ebene definiert haben, gilt folgender Zusammenhang: $\theta = 90° - \alpha$. Da nur Orientierungen betrachtet werden, kann für den Radius r der Kugel $r = 1$ gesetzt werden.

Einsetzen in die allgemeinen Formeln 4.1 liefert die endgültigen konkreten Gleichungen, um PLI-Kugelkoordinaten in kartesische Koordinaten umzurechnen:

$$\begin{aligned} x &= \sin(90° - \alpha) \cdot \cos(\delta) = \cos(\alpha) \cdot \cos(\delta) \\ y &= \sin(90° - \alpha) \cdot \sin(\delta) = \cos(\alpha) \cdot \sin(\delta) \\ z &= \cos(90° - \alpha) \qquad\quad = \sin(\alpha) \end{aligned} \tag{4.2}$$

Es ist wichtig, den Wertebereich der PLI-Kugelkoordinaten zu kennen. Für den Wertebereich des PLI-Workflows gilt $\delta \in [0; 180]$ und $\alpha \in [-90; 90]$. Nach Umrechnung in allgemeine Kugelkoordinaten zeigt sich, dass wegen $\phi \in [0; 180]$ und $\theta \in [0; 180]$ nur Koordinaten mit positiver y-Koordinate erreicht werden können.

Das bedeutet, dass bei der PLI-Aufnahme zweier Fasern bzw. Vektoren mit Richtungen (1, 1, 1) und (-1, -1, -1) dieselben Winkel gemessen werden. Die beiden Fälle sind als PLI-Kugelkoordinaten nicht voneinander unterscheidbar.

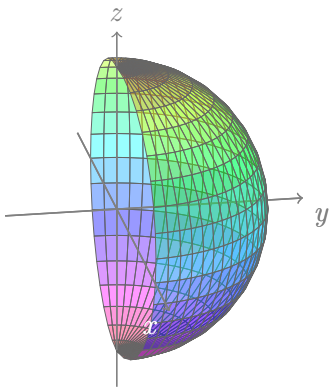


Abbildung 4.2: Wertebereich der Kugelkoordinaten

4.3 Skalenüberbrückung durch Vektorinterpolation

Die einfachste Methode um Richtungsinformationen von mehreren Bildpunkten zusammenzufassen, ist das Bilden der Mittelrichtung der Vektoren. Alternativ könnte man auch die Mittelwerte der Winkel bilden. Diese besitzen jedoch keine linearen Zusammenhänge und müssten daher mit den in Kapitel 4.2 genannten Formeln 4.2 in Vektoren transformiert werden.

Bei der Verarbeitung der Vektoren ist zu beachten, dass nur der halbe Wertebereich abgedeckt wird (siehe Abb. 4.2). Aus diesem Grund kann die Hauptrichtung zweier Werte in vielen Fällen nicht durch das arithmetische Mittel bestimmt werden.

So sieht man beispielsweise in Abbildung 4.3, dass der direkt berechnete Mittelwert von $\overrightarrow{v_1}$ und $\overrightarrow{v_2}$ der rot dargestellte Vektor $\overrightarrow{v_3}$ ist. Da die Richtung eines PLI-Vektors unbekannt ist, könnte z.B. $\overrightarrow{v_2}$ in der Realität wie $\overrightarrow{v_2^*}$ verlaufen. Der als Mittelwert von $\overrightarrow{v_1}$ und $\overrightarrow{v_2^*}$ gebildetete Vektor $\overrightarrow{v_3^*}$ verläuft senkrecht zum Vektor $\overrightarrow{v_3}$.

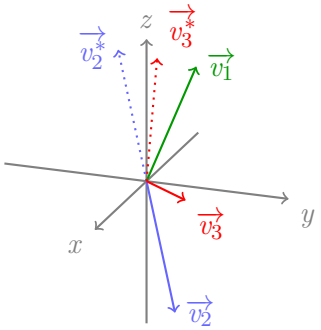


Abbildung 4.3: Beispiel für verschiedene mögliche Mittelwerte

Beweis. Als normierte Vektoren gegeben sind

$$\overrightarrow{v_1} = \begin{pmatrix} \frac{a}{\sqrt{a^2+b^2+c^2}} \\ \frac{b}{\sqrt{a^2+b^2+c^2}} \\ \frac{c}{\sqrt{a^2+b^2+c^2}} \end{pmatrix} \text{ und } \overrightarrow{v_2} = \begin{pmatrix} \frac{d}{\sqrt{d^2+e^2+f^2}} \\ \frac{e}{\sqrt{d^2+e^2+f^2}} \\ \frac{f}{\sqrt{d^2+e^2+f^2}} \end{pmatrix}$$

Als Mittelvektor ergibt sich

$$\overrightarrow{v_3} = \frac{\overrightarrow{v_1}+\overrightarrow{v_2}}{2} = \begin{pmatrix} \frac{a\cdot\sqrt{d^2+e^2+f^2} \; + \; d\cdot\sqrt{a^2+b^2+c^2}}{2\cdot\sqrt{a^2+b^2+c^2}\cdot\sqrt{d^2+e^2+f^2}} \\ \frac{b\cdot\sqrt{d^2+e^2+f^2} \; + \; e\cdot\sqrt{a^2+b^2+c^2}}{2\cdot\sqrt{a^2+b^2+c^2}\cdot\sqrt{d^2+e^2+f^2}} \\ \frac{c\cdot\sqrt{d^2+e^2+f^2} \; + \; f\cdot\sqrt{a^2+b^2+c^2}}{2\cdot\sqrt{a^2+b^2+c^2}\cdot\sqrt{d^2+e^2+f^2}} \end{pmatrix}$$

Invertiert man $\overrightarrow{v_2}$ erhält man den Vektor $\overrightarrow{v_2^*}$ und den neuen Mittelvektor $\overrightarrow{v_3^*}$

$$\overrightarrow{v_2^*} = \begin{pmatrix} \frac{-d}{\sqrt{d^2+e^2+f^2}} \\ \frac{-e}{\sqrt{d^2+e^2+f^2}} \\ \frac{-f}{\sqrt{d^2+e^2+f^2}} \end{pmatrix}, \quad \overrightarrow{v_3^*} = \frac{\overrightarrow{v_1}+\overrightarrow{v_2^*}}{2} = \begin{pmatrix} \frac{a\cdot\sqrt{d^2+e^2+f^2} \; - \; d\cdot\sqrt{a^2+b^2+c^2}}{2\cdot\sqrt{a^2+b^2+c^2}\cdot\sqrt{d^2+e^2+f^2}} \\ \frac{b\cdot\sqrt{d^2+e^2+f^2} \; - \; e\cdot\sqrt{a^2+b^2+c^2}}{2\cdot\sqrt{a^2+b^2+c^2}\cdot\sqrt{d^2+e^2+f^2}} \\ \frac{c\cdot\sqrt{d^2+e^2+f^2} \; - \; f\cdot\sqrt{a^2+b^2+c^2}}{2\cdot\sqrt{a^2+b^2+c^2}\cdot\sqrt{d^2+e^2+f^2}} \end{pmatrix}$$

Berechnen des Skalarproduktes von $\overrightarrow{v_3}$ und $\overrightarrow{v_3^*}$ als Beweis der Orthogonalität:

$$\begin{aligned} \langle \overrightarrow{v_3}, \overrightarrow{v_3^*} \rangle &= \frac{a^2\cdot\left(d^2+e^2+f^2\right) \; - \; d^2\cdot\left(a^2+b^2+c^2\right)}{4\cdot(a^2+b^2+c^2)\cdot(d^2+e^2+f^2)} + \frac{b^2\cdot\left(d^2+e^2+f^2\right) \; - \; e^2\cdot\left(a^2+b^2+c^2\right)}{4\cdot(a^2+b^2+c^2)\cdot(d^2+e^2+f^2)} \\ &\quad + \frac{c^2\cdot\left(d^2+e^2+f^2\right) \; - \; f^2\cdot\left(a^2+b^2+c^2\right)}{4\cdot(a^2+b^2+c^2)\cdot(d^2+e^2+f^2)} \\ &= \frac{\left(a^2+b^2+c^2\right)\cdot\left(d^2+e^2+f^2\right) \; - \; \left(d^2+e^2+f^2\right)\cdot\left(a^2+b^2+c^2\right)}{4\cdot(a^2+b^2+c^2)\cdot(d^2+e^2+f^2)} \\ &= 0 \end{aligned}$$

□

An dieser Stelle muss entschieden werden, welcher dieser Vektoren das richtige Ergebnis darstellt. Andernfalls wären die Ergebnisse der Interpolation vollkommen unbrauchbar für weitere Berechnungen, da beliebig starke Abweichungen von der Realität vorliegen würden.

Um entscheiden zu können, welcher der beiden Mittelvektoren $\overrightarrow{v_3}$ und $\overrightarrow{v_3^*}$ der richtige ist, müssen Kriterien entworfen werden, anhand derer man diese Entscheidung treffen kann. So ist z.B. davon auszugehen, dass benachbarte Pixel in den meisten Fällen ähnliche Richtungsinformationen enthalten, da sich Faserorientierungen *meistens* nicht sprunghaft ändern. Das bedeutet, dass die Vektoren vor dem Bilden der Mittelrichtung immer so am Ursprung gespiegelt (invertiert) werden sollten, dass sie einander am ähnlichsten sind. Im obigen Fall wären das $\overrightarrow{v_1}$ und $\overrightarrow{v_2^*}$ anstelle von $\overrightarrow{v_2}$. Ein einfaches Kriterium für ähnliche Richtungen ist das Skalarprodukt. Je größer es ist, desto ähnlicher sind die Richtungen der beiden Vektoren. Zusätzlich bedeutet ein Skalarprodukt $\langle a, b\rangle > 0$, dass $\measuredangle a, b \in [0°;\ 90°)$. Ebenso gilt $\langle a, b\rangle < 0 \Rightarrow\ \measuredangle a, b \in (90°;\ 180°]$.

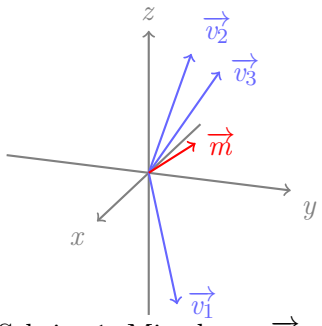
4.3.1 Methode 1: *Step-by-Step*

Um die korrekte Mittelrichtung zu erhalten, müssen in manchen Fällen Vektoren invertiert werden. Dabei trifft man auf ein weiteres Problem. Es stellt sich die Frage, wie man für eine größere Anzahl von Vektoren vorgeht. Man müsste beispielsweise 1000 Vektoren verarbeiten bei einer Verkleinerung des Volumenbildes um den Faktor 10 in alle Richtungen. Das Ausprobieren aller Möglichkeiten der Invertierung ist nicht möglich, da pro Pixel $2^{1000} \approx 1,07 \cdot 10^{301}$ Kombinationen (mathematisch: Permutation mit Wiederholung) von invertierten und nicht invertierten Vektoren berechnet werden müssten. Selbst bei einer Verkleinerung um Faktor 3 in alle Richtungen wären es noch $2^{27} = 134.217.728$ zu testende Kombinationen.

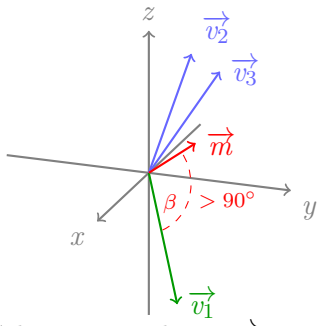
Aus diesem Grund arbeitet die *Step-by-Step*-Methode die Vektoren jeweils sequentiell ab. Idee dabei ist, die Mittelrichtung Stück für Stück aufzubauen und die einzelnen Vektoren bei Bedarf zu invertieren.

Algorithmus

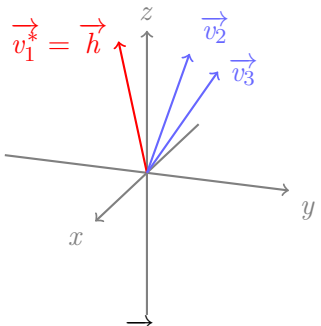
Im ersten Schritt des Algorithmus wird der Durchschnittsvektor (ohne Invertierung) aller Vektoren gebildet. Ist der Winkel β zwischen diesem Durchschnittsvektor und dem ersten Eintrag des Vektorfeldes größer als 90°, so wird der erste Eintrag invertiert. Dieser erste Eintrag wird als neue temporäre Hauptrichtung gespeichert.



Schritt 1: Mittelwert $\overrightarrow{m}$ aller Vektoren bilden



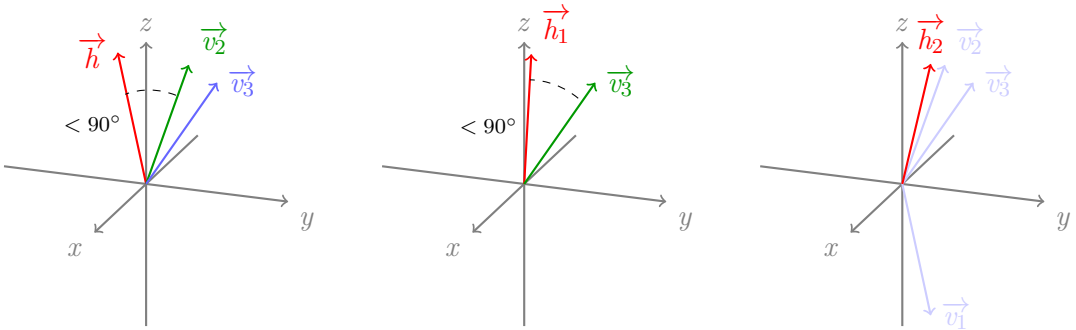
Schritt 2: Vektor $\overrightarrow{v_1}$ invertieren, weil Winkel zur Mittelrichtung $> 90°$



Schritt 3: $\overrightarrow{v_1^*}$ als temporäre Hauptrichtung $\overrightarrow{h}$ übernehmen

Abbildung 4.4: Die ersten Schritte der *Step-by-Step*-Methode

Alle weiteren Einträge des Vektorfeldes werden folgendermaßen behandelt: Ist der Winkel zwischen Eintrag und aktueller Hauptrichtung größer als 90°, so wird der Eintrag invertiert. Die neue Hauptrichtung wird bestimmt durch den normierten Mittelvektor aus alter Hauptrichtung und eventuell invertiertem Eintrag. Dabei wird die alte Hauptrichtung um einen Faktor gewichtet, der der Anzahl der bisher verarbeiteten Einträge der Liste entspricht.



Schritt 4: Vektor $\overrightarrow{v_2}$ nicht invertieren, weil Winkel zur aktuellen Hauptrichtung < 90°

Schritt 5: Bilde Mittelrichtung $\overrightarrow{h_1}$ aus $\overrightarrow{h}$ und $\overrightarrow{v_2}$ und prüfe, ob Winkel zu $\overrightarrow{v_3}$ < 90°

Schritt 6: Bilde Mittelrichtung $\overrightarrow{h_2}$ aus $\overrightarrow{h_1}$ und $\overrightarrow{v_3}$ mit doppelter Gewichtung von $\overrightarrow{h_1}$

Abbildung 4.5: Die weiteren Schritte der *Step-by-Step*-Methode

Vektor **bestimmeHauptrichtung**(Vektor[] `feld`)

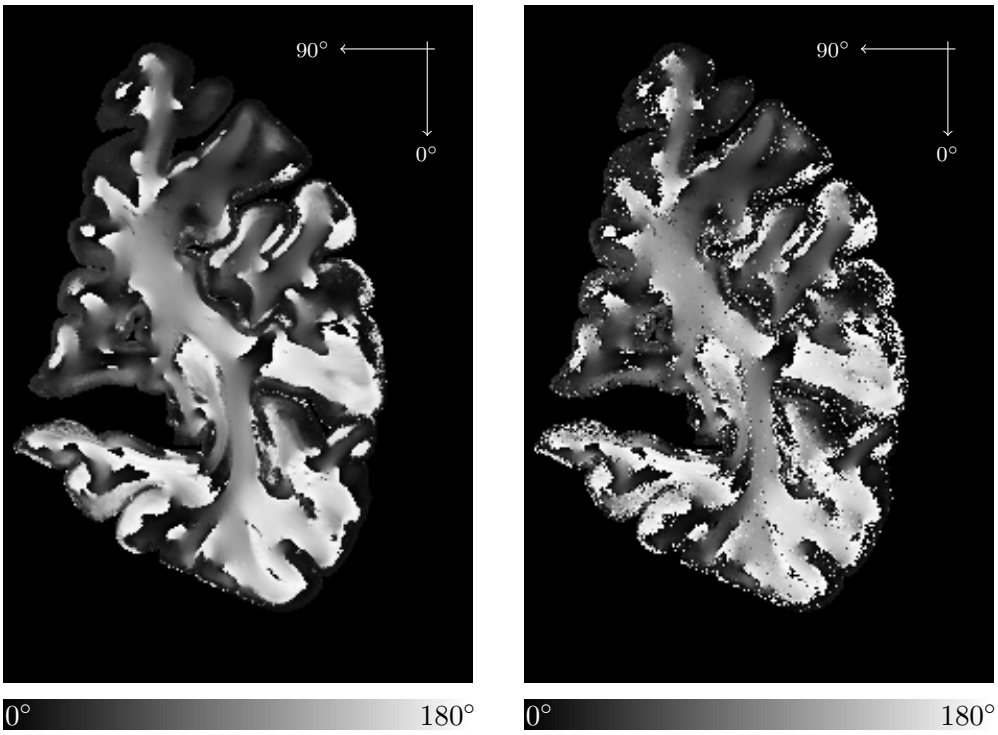
Bestimme den Mittelwert der Vektoren als Hauptrichtung $\overrightarrow{h}$
Invertiere `feld[0]`, wenn der Winkel zu $\overrightarrow{h}$ größer ist als 90°
Setze `feld[0]` als neue Hauptrichtung $\overrightarrow{h}$
Für alle weiteren Indices `i`
Invertiere `feld[i]`, wenn der Winkel zu $\overrightarrow{h}$ größer ist als 90°
Bilde neue gewichtete Mittelrichtung aus Hauptrichtung und `feld[i]` als $\overrightarrow{h} = \left(\mathtt{i} \cdot \overrightarrow{h} + \mathtt{feld[i]}\right) / (\mathtt{i} + 1)$
Normiere $\overrightarrow{h}$, da mit Richtungen gearbeitet wird
$\overrightarrow{h}$ enthält die Mittelrichtung der Vektoren

Nassi 4.1: Algorithmus der *Step-by-Step*-Methode

Auswertung - Hindernisse/Probleme

Vergleicht man das Ergebnisbild dieser Methode bei Verkleinerung um den Faktor 6 in der Schnittebene mit dem Bild, das durch die einfache Mittelrichtung der Vektoren erzeugt wurde, sieht man bezüglich der Glattheit der Bilder starke Unterschiede. Das Bild der Mittelrichtungen enthält im Vergleich viel weniger Rauschen.

Die einfache Mittelrichtung erhält man, wenn man die Summe der für ein Pixel zu verarbeitenden Vektoren bildet und diese mit der euklidischen Norm normiert.



(a) einfache Mittelrichtung (b) *Step-by-Step*-Methode

Abbildung 4.6: Ergebnisbilder der einfachen Mittelrichtung und der *Step-by-Step*-Methode bei Verkleinerung des Originalbildes um den Faktor $6 \times 6 \times 1$. Dieser Faktor bedeutet eine Verkleinerung um den Faktor 6 in beiden Dimensionen eines Schnittes (x- und y-Richtung des Volumenbildes) und eine Verkleinerung um den Faktor 1 in z-Richtung, d.h. über mehrere Schnitte hinweg. Gezeigt ist stellvertretend das Bild der Direktionswinkel.

Bleibt zu klären, inwiefern das Ergebnisbild korrekt ist und an welchen Stellen Probleme aufgetreten sein könnten. Die einzige Stelle des Algorithmus, die von der Berechnung eines einfachen Mittelvektors abweicht, ist die Stelle, an der Vektoren invertiert werden. Invertiert wird, wenn das Skalarprodukt kleiner als Null ist, d.h. der Winkel zwischen Mittelrichtung und aktuellem Vektor ist größer als 90 Grad.

Ein Skalarprodukt von etwa Null, d.h. ein Winkel zwischen 85 und 95 Grad, bedeutet, dass nicht mit Sicherheit entschieden werden kann, ob der Vektor invertiert werden muss. Während eine Invertierung auf den Winkel zwischen aktueller Mittelrichtung und zu betrachtendem Vektor kaum Einfluss hat, ändert sich bei einer Invertierung jedoch die neue Mittelrichtung.

Wie in Tabelle 4.1 abzulesen, tritt unter den gegebenen Umständen in etwa einem von fünf Fällen mindestens einmal ein solches Problem auf. Dieser Wert ist jedoch selbst auch wieder zufällig, da eine andere Bearbeitungsreihenfolge der selben Vektoren auch andere Winkel zwischen temporärer Mittelrichtung und Vektor liefert. Aus diesem Grund ist das Ergebnis bei dieser Methode abhängig von der Reihenfolge, in der die Vektoren verarbeitet werden.

Anzahl der rechten Winkel	absolute Häufigkeit	relative Häufigkeit
0	4250578	80,656%
1	521984	9,905%
2	263681	5,003%
3	125779	2,387%
4	59627	1,131%
5	27249	0,517%
6	12052	0,229%
>6	9090	0,172%

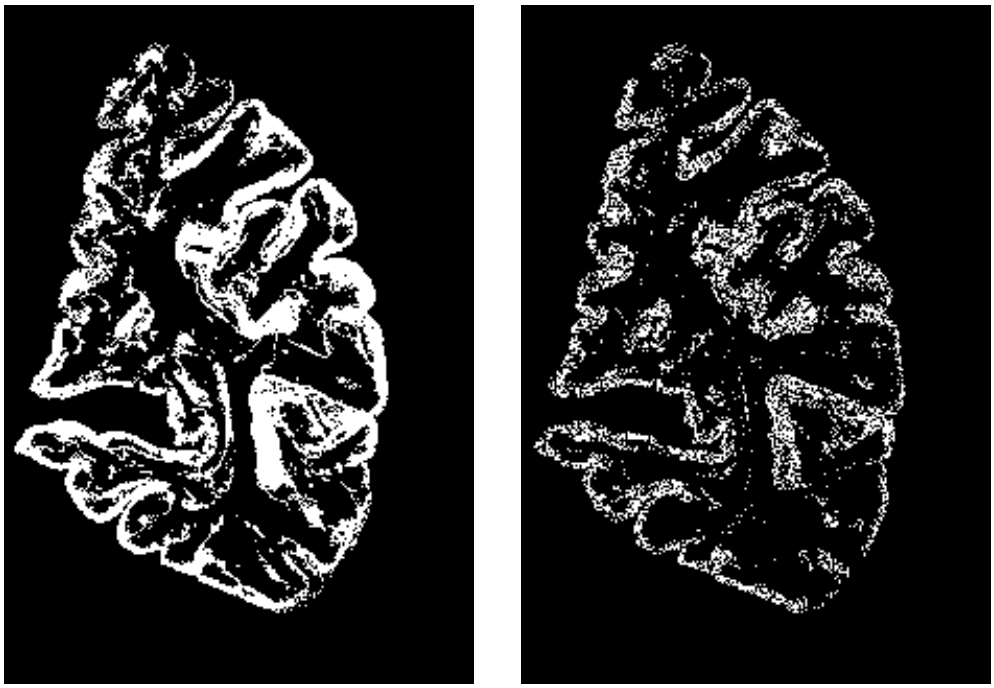
Tabelle 4.1: Häufigkeit des Auftretens von nahezu rechten Winkeln (85-95 Grad) zwischen aktueller Mittelrichtung und betrachtetem Vektor bei der *Step-by-Step*-Methode mit Verkleinerung um den Faktor $6 \times 6 \times 1$

Erzeugt man das Differenzbild von Abb. 4.6(a) und Abb. 4.6(b) und markiert alle Pixel, bei denen die Summe der Beträge der Abweichungen in Direktion bzw. Inklination mehr als 5 Grad beträgt, sieht man, an welcher Stelle die Methode abweichende Ergebnisse erzeugt. Dieses Bild kann man mit dem Bild vergleichen, in dem alle Pixel markiert sind, bei deren Berechnung mindestens ein rechter Winkel aufgetreten ist.

Es ist deutlich zu erkennen, dass sowohl Abweichungen als auch Probleme bis auf einige wenige Ausnahmen ausschließlich in der grauen Substanz auftreten. Das liegt zum einen an der Homogenität der weißen Substanz, die eine Invertierung der Vektoren fast immer unnötig macht, aber auch an der geringeren Myelindichte der grauen Substanz. Aus dieser folgt eine geringe Signalamplitude und daher auch ein geringer Retardierungswert. Dadurch sind diese Bereiche wesentlich rauschanfälliger.

Es fällt auf, dass der Großteil der Pixel, deren Richtung von der einfachen Mittelrichtung um mehr als 5 Grad abweicht, ebenfalls zu den Pixeln gehört, deren Richtung nicht zweifelsfrei bestimmt werden konnte.

Da das Auftreten von rechten Winkeln bei der Berechnung nicht vorhergesagt oder vermieden werden kann und zusätzlich von der Bearbeitungsreihenfolge der Vektoren abhängt, muss diese Methode als instabil und unzuverlässig eingestuft werden.



(a) Abweichung zur Mittelrichtung $> 5°$ (b) Pixel ohne eindeutige Richtung

Abbildung 4.7: Auswertung des Unterschiedes von einfacher Mittelrichtung und *Step-by-Step*-Methode durch weiße Markierung aller Pixel, deren Richtungsdifferenz größer als 5 Grad ist. Dabei wird die Summe der Differenzen in Direktion und Inklination betrachtet. Dem gegenüber sind alle Pixel ohne eindeutige Richtung, d.h. während der Berechnung auftretender rechter Winkel, ebenfalls weiß markiert.

4.3.2 Methode 2: *Flipping-the-worst*

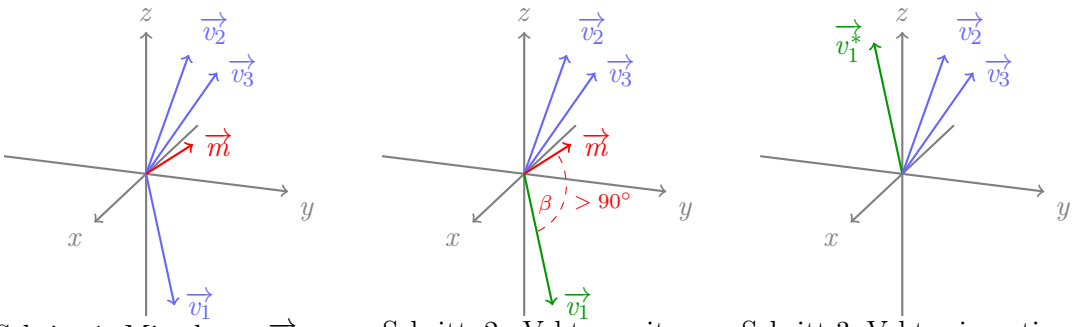
Bei der *Step-by-Step*-Methode beeinflusst die Bearbeitungsreihenfolge das Ergebnis. Um eine stabilere Methode zu erhalten, darf die Bearbeitungsreihenfolge keinen Einfluss auf das Ergebnis haben. Das bedeutet, dass es keine temporäre Mittelrichtung geben darf, die als Zwischenergebnis für einen weiteren Rechenschritt dient. Gleichzeitig müssen aber nach wie vor Vektoren invertiert werden, um das bestmögliche Ergebnis bei der Berechnung der Hauptrichtung zu erhalten.

Die *Flipping-the-worst*-Methode wurde so gestaltet, dass die temporäre Mittelrichtung nach jeder Invertierung verworfen und neu berechnet wird.

Algorithmus

Im ersten Schritt des Algorithmus wird, wie auch bei der *Step-by-Step*-Methode, der Durchschnittsvektor aller Richtungen gebildet. Danach werden alle Skalarprodukte der

Vektoren mit diesem Durchschnittsvektor gebildet. Der Vektor, bei dem das Skalarprodukt am kleinsten ist, d.h. der den größten Winkel zur Mittelrichtung aufweist, wird invertiert, wenn der Winkel größer als 90 Grad ist.



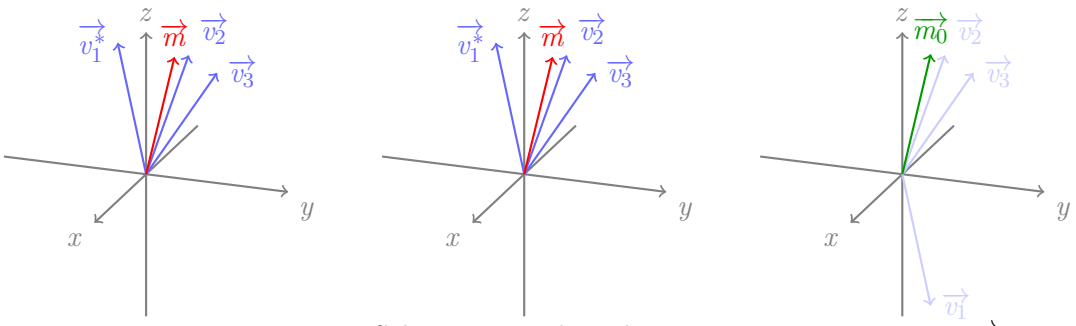
Schritt 1: Mittelwert $\overrightarrow{m}$ aller Vektoren bilden

Schritt 2: Vektor mit größtem Winkel finden

Schritt 3: Vektor invertieren, da Winkel $\beta > 90°$

Abbildung 4.8: Die ersten Schritte der *Flipping-the-worst*-Methode

Nach der Invertierung wird die neue Mittelrichtung berechnet und wieder der Vektor mit dem größten Winkel zur Mittelrichtung invertiert, sofern dieser größer als 90 Grad ist. Dieser Vorgang wird wiederholt, bis alle Winkel kleiner sind als 90 Grad. Die dann berechnete Mittelrichtung wird normiert und als Hauptrichtung angesehen.



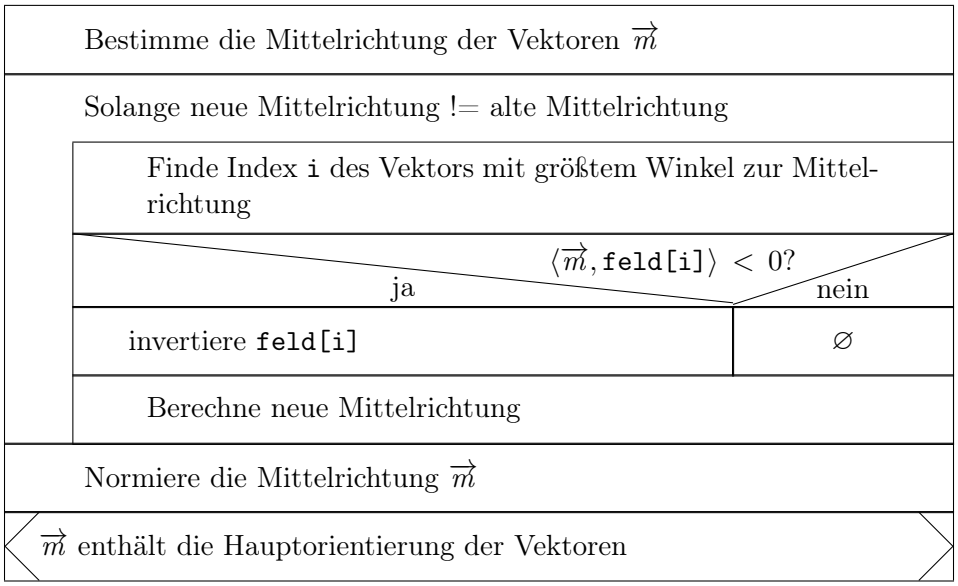
Schritt 4: Bilde neuen Mittelwert $\overrightarrow{m}$ aller Vektoren

Schritt 5: Finde Vektor mit größtem Winkel, für den $\beta > 90°$ gilt.

Schritt 6: Gib $\overrightarrow{m_0}$ als normierte Ergebnisrichtung zurück, da alle Winkel kleiner als 90 Grad sind.

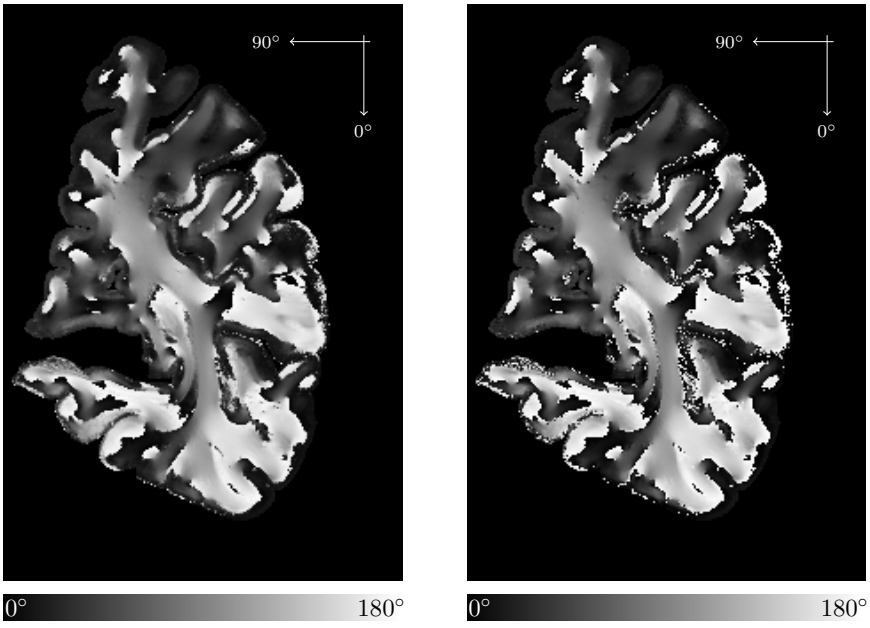
Abbildung 4.9: Die weiteren Schritte der *Flipping-the-worst*-Methode

Vektor **bestimmeHauptrichtung**(Vektor[] `feld`)



Nassi 4.2: Algorithmus der *Flipping-the-worst*-Methode

Auswertung - Hindernisse/Probleme



(a) einfache Mittelrichtung (b) *Flipping-the-worst*-Methode

Abbildung 4.10: Ergebnisbilder (Direktion) der einfachen Mittelrichtung und der *Flipping-the-worst*-Methode bei Verkleinerung des Originalbildes um den Faktor $6 \times 6 \times 1$.

Im Gegensatz zum Ergebnisbild der *Step-by-Step*-Methode (Abb. 4.6(b)) sieht man bei dieser Methode nur leichte Unterschiede zur einfachen Mittelrichtung der Vektoren (Abb. 4.10). Nur an einigen Stellen der grauen Substanz, insbesondere im Cortex, der Hirnrinde, ist ein Rauschen erkennbar.

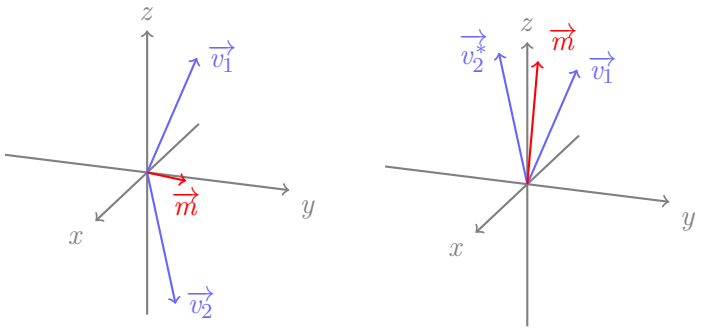
Die Ergebnisse einer Methode können durch Eindeutigkeit bestätigt werden. Dafür muss die Anzahl der aufgetretenen Problemfälle analysiert werden. Bei einer genaueren Untersuchung des Algorithmus fällt auf, dass der Problemfall rechter Winkel ausschließlich am Ende der Berechnungen auftreten kann. Bevor die Schleife zur Invertierung des schlechtesten Vektors abbricht, sind alle betrachteten Winkel kleiner oder gleich 90 Grad.

Als Problem zählt demnach ein rechter Winkel zwischen Ergebnisrichtung und einem der Vektoren. Daher ist die Anzahl der Probleme unabhängig von der Bearbeitungsreihenfolge, was als Ziel der Methode definiert war.

Anzahl der rechten Winkel	absolute Häufigkeit	relative Häufigkeit
0	4702818	89,237%
1	361947	6,868%
2	136372	2,588%
3	45458	0,863%
>3	23445	0,445%

Tabelle 4.2: Häufigkeit des Auftretens von nahezu rechten Winkeln (85-95 Grad) zwischen finaler Mittelrichtung und einzelnen Vektoren bei *Flipping-the-worst* mit Verkleinerung um den Faktor 6×6×1 ert

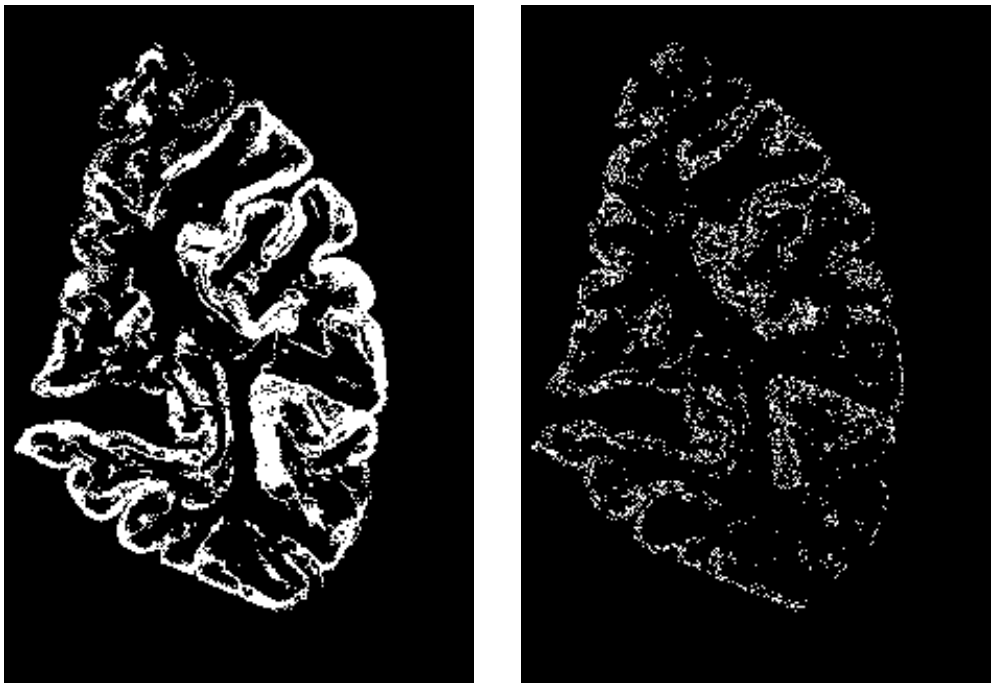
Vergleicht man die Häufigkeit der Problemfälle der *Step-by-Step*-Methode (Tab. 4.1 auf Seite 18) mit der der *Flipping-the-worst*-Methode (Tab. 4.2), sieht man, dass sich die Anzahl der Problemfälle etwa halbiert hat. Das bedeutet, dass die Ergebnisse eindeutiger geworden sind, macht über die Korrektheit der Ergebnisse jedoch keine Aussage.



(a) ohne Invertierung (b) mit Invertierung

Abbildung 4.11: Die Vektoren der linken Skizze würde der Algorithmus *Flipping-the-worst* nicht invertieren, da die Winkel zur Mittelrichtung kleiner sind als 90°. Offensichtlich wäre das Ergebnis bei einer Invertierung besser (rechte Skizze).

Eine genauere Betrachtung des Algorithmus zeigt, dass die Methode zum Teil falsche Ergebnisse liefert. So wird z.B. bei einer Konstellation wie in Abbildung 4.11 keiner der Vektoren invertiert, weil der Winkel zur Mittelrichtung bei beiden kleiner ist als 90 Grad, obwohl die Vektoren offensichtlich besser zueinander lägen, würde einer von ihnen invertiert.



(a) Abweichung zur Mittelrichtung $> 5°$ (b) Pixel ohne eindeutige Richtung

Abbildung 4.12: Auswertung des Unterschiedes von einfacher Mittelrichtung und *Flipping-the-worst*-Methode durch weiße Markierung aller Pixel, deren Richtungsdifferenz größer als 5 Grad ist. Dabei wird die Summe der Differenzen in Direktion und Inklination betrachtet. Dem gegenüber sind alle Pixel ohne eindeutige Richtung, d.h. nach der Berechnung noch vorhandener rechter Winkel, ebenfalls weiß markiert.

Eine Betrachtung der Lage der Pixel, nach deren Berechnung rechte Winkel auftreten, zeigt auch bei dieser Methode, dass sie nahezu ausschließlich in der grauen Substanz liegen. Allerdings zeigt die Grafik der Pixel mit Abweichung zur Mittelrichtungsmethode im Vergleich zu den Pixeln, deren Ergebnisrichtung nicht eindeutig ist, dass ein Teil der abweichenden Pixel keine zweite mögliche Ergebnisrichtung hat.

Das angewandte Kriterium garantiert, dass alle Vektoren in einer Halbkugel um den Mittelvektor liegen, weil alle Winkel spitz sind. Dadurch ist es zur Korrektur einzelner Ausreißer geeignet, aber nicht zum Finden der besten Lösung. Aus diesem Grund ist das Verfahren ebenfalls unbrauchbar und muss verworfen werden.

4.3.3 Methode 3: *Cluster-Flipping*

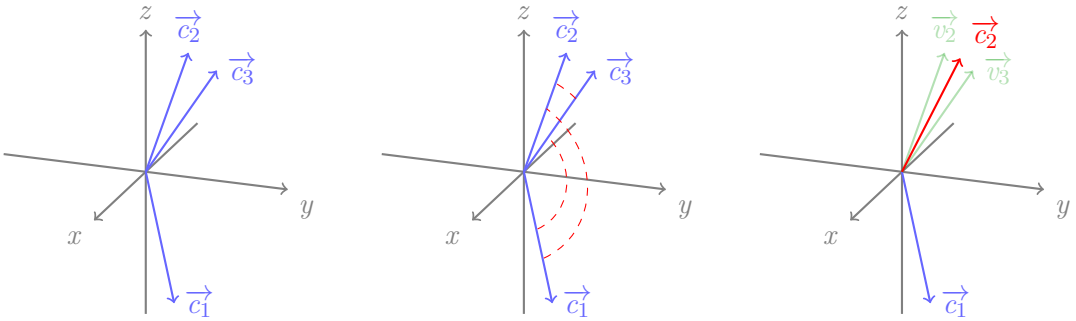
Eine Methode zur Zusammenfassung von Vektoren sollte auch rechte Winkel annähernd korrekt verarbeiten können. Dafür muss das Entscheidungskriterium angepasst werden. Die ideale Lösung, bei der die Winkel zwischen den Vektoren kleinstmöglich bzw. die Skalarprodukte größtmöglich sind, wird nur gefunden, wenn man alle Möglichkeiten ausprobiert.

Da aus den in Kapitel 4.3.1 auf Seite 15 erwähnten Gründen ein Testen aller Kombinationen unmöglich ist, muss versucht werden, die Anzahl der Kombinationen zu verringern. Dazu kann man Vektoren mit ähnlicher Richtung in Gruppen zusammenzufassen. Dadurch reicht es, alle Kombinationen der Gruppen-Invertierungen zu testen, weil ähnliche Vektoren normalerweise nur gemeinsam gespiegelt werden. Andernfalls würde man erreichen, dass zwei ähnliche Vektoren sich im Ergebnis aufheben und nicht verstärken. Begrenzt man die Gruppenanzahl auf 4, dann müssen nur noch $2^4 = 16$ Kombinationen getestet werden, was in annehmbarer Zeit möglich ist.

Algorithmus

Der Algorithmus selbst ist in zwei Abschnitte unterteilt. Der erste Abschnitt ist das Bilden der Gruppen und der zweite Abschnitt das Finden der bestmöglichen Kombination der Gruppeninvertierungen.

Die Vektoren werden dabei jeweils in eine eigene Gruppe eingeteilt. Wird dabei die maximale Anzahl an Gruppen um eins überschritten, müssen zwei Gruppen zusammengefasst werden. An dieser Stelle greift wieder das Kriterium des größten Skalarproduktes. Man berechnet alle Skalarprodukte zwischen jeweils zwei Gruppen und fasst die beiden Gruppen mit dem größten Skalarprodukt, d.h. mit dem kleinsten Winkel, zusammen. Nach dem Zusammenfassen errechnet man für die Gruppe die normierte Mittelrichtung als Hauptorientierung. Der Vorgang der Gruppenerzeugung und des Zusammenfassens wird so lange wiederholt, bis alle Vektoren in Gruppen eingeteilt sind.



Schritt 1a: Alle Vektoren je einer Gruppe zuweisen

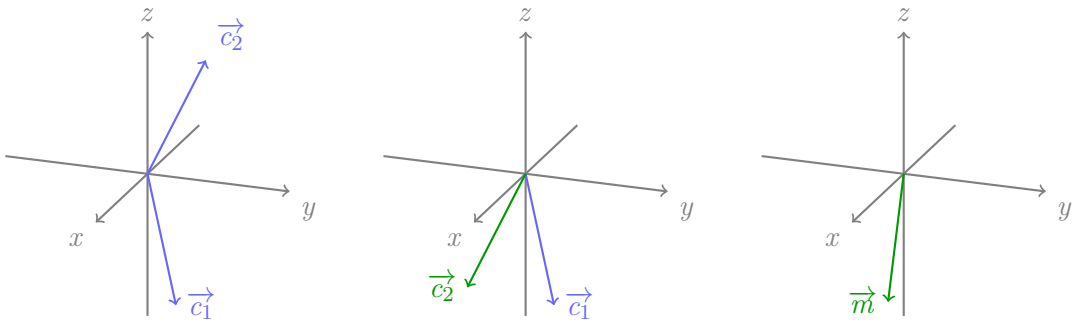
Schritt 1b: Alle Winkel, bzw. Skalarprodukte berechnen

Schritt 1c: Die Gruppen mit dem kleinsten Winkel zusammenfassen.

Abbildung 4.13: Einteilung der Gruppen bei der *Cluster-Flipping*-Methode. In diesem Beispiel werden drei Vektoren verarbeitet, das Gruppenlimit liegt bei zwei Gruppen.

Nach der Einteilung in Gruppen müssen nur noch alle Kombinationen von Gruppeninvertierungen ausprobiert werden. Invertiert man alle Gruppen gleichzeitig, so dreht sich der Ergebnisvektor um. Er hat jedoch die gleiche Orientierung, weil das Vorzeichen als unbekannt angesehen werden muss. Daher muss eine Gruppe nicht invertiert werden. Für N Gruppen müssen also anstelle von 2^N nur 2^{N-1} Kombinationen überprüft werden. Bei der Implementierung wurde $N = 5$ gewählt, da $2^4 = 16$ Kombinationen in annehmbarer Zeit berechnet werden können.

Um der unterschiedlichen Anzahl an Vektoren pro Cluster eine Gewichtung zu verleihen, wird nicht das mittlere Skalarprodukt der Hauptrichtungen der Cluster verglichen, sondern die Norm der Summe der Vektoren aller Cluster. Ergebnisrichtung ist der normierte Durchschnittsvektor aller Vektoren bei der Kombination mit der größten Norm.



Schritt 2a: Berechnen der Norm der summierten Vektoren der Cluster

Schritt 2b: Invertieren des zweiten Clusters, berechnen der neuen Norm

Schritt 2c: Berechnen der Mittelrichtung $\overrightarrow{m}$ für die Kombination mit der größten Norm

Abbildung 4.14: Testen der Kombinationen bei der *Cluster-Flipping*-Methode

Vektor **bestimmeHauptrichtung**(Cluster[] `cluster`)

Für jede Kombination von Invertierungen der Cluster 2 bis 5

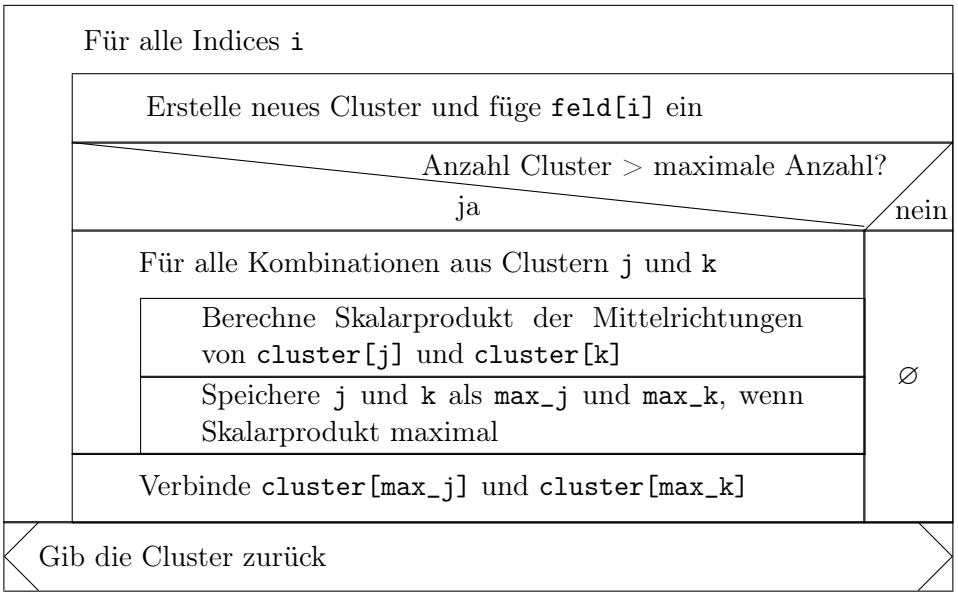
`tmpVal` = Norm der Summe der Vektoren aller Cluster

Speichere die Kombination der Invertierungen wenn `tmpVal` maximal

Gib die normierte Mittelrichtung aller Vektoren der gespeicherten Kombination zurück

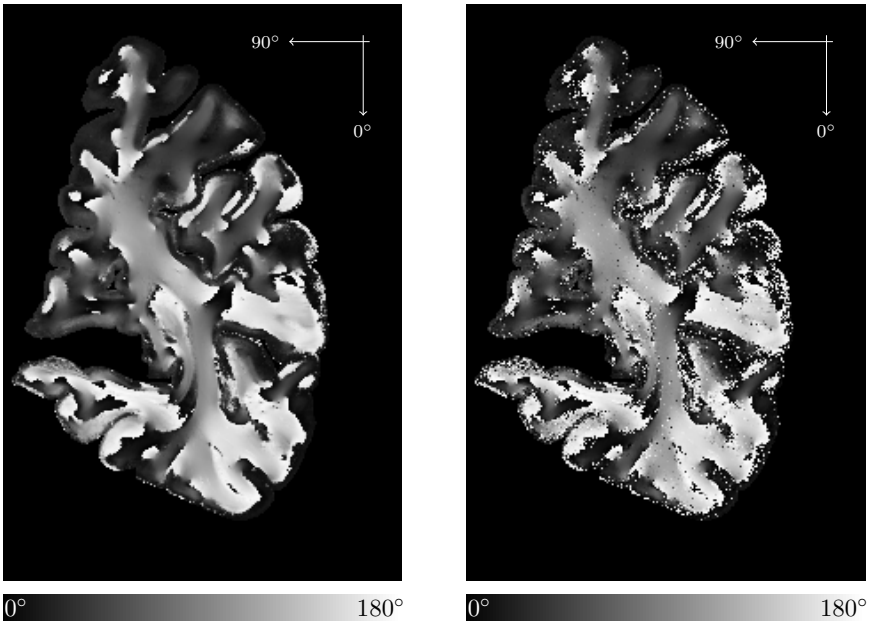
Nassi 4.3: Bestimmen der Hauptrichtung bei der *Cluster-Flipping*-Methode

Cluster[] **erstelleCluster**(Vektor[] `feld`)



Nassi 4.4: Erstellen der Gruppen bei der *Cluster-Flipping*-Methode

Auswertung - Hindernisse/Probleme



(a) einfache Mittelrichtung (b) *Cluster-Flipping*-Methode

Abbildung 4.15: Ergebnisbilder (Direktion) der einfachen Mittelrichtung und der *Cluster-Flipping*-Methode bei Verkleinerung des Originalbildes um den Faktor $6 \times 6 \times 1$.

Wie bei allen bisher überprüften Methoden sieht man einen Unterschied zwischen den Mittelrichtungen (Abb. 4.15(a)) und dem berechneten Bild (Abb. 4.15(b)). Dabei fällt auf, dass das Rauschen im Cortex wieder zugenommen hat, sodass das Bild wieder Ähnlichkeiten mit dem Ergebnisbild der ersten Methode (Abb. 4.6(b) auf Seite 17) aufweist.

Um eine Aussage über die Güte der Ergebnisse treffen zu können, muss eine weitere Betrachtung der Problemfälle vorgenommen werden. Da sich das Bewertungskriterium der Lösungen grundlegend geändert hat, muss sich zwangsläufig auch die Untersuchung von Problemen ändern. Das Vergleichskriterium ist bei der *Cluster-Flipping*-Methode die Norm des Vektors, den man erhält, wenn man die Vektoren aller Cluster addiert.

Das bedeutet, dass ein Problem dann vorliegt, wenn zwei Normen ähnlich sind und sich die Ergebnisrichtungen unterscheiden. Gewählte Grenze dabei ist mehr als 10° Unterschied der Richtungen bei weniger als 1% Unterschied zwischen der Norm einer Kombination und der Norm der als Ergebnis gewählten Kombination.

(a) Abweichung zur Mittelrichtung $> 5°$ (b) Pixel ohne eindeutige Richtung

Abbildung 4.16: Auswertung des Unterschiedes von einfacher Mittelrichtung und *Cluster-Flipping*-Methode durch weiße Markierung aller Pixel, deren Richtungsdifferenz größer als 5 Grad ist. Dabei wird die Summe der Differenzen in Direktion und Inklination betrachtet. Dem gegenüber sind alle Pixel ohne eindeutige Richtung, d.h. ähnlicher Norm und abweichender Ergebnisrichtung bei Auswertung einer Kombination, ebenfalls weiß markiert.

Anzahl zusätzlicher Richtungen	absolute Häufigkeit	relative Häufigkeit
0	2405795	45,777%
1	1761901	33,525%
2	605168	11,515%
3	168008	3,179%
4	139659	2,657%
>4	189509	3,596%

Tabelle 4.3: Häufigkeit des Auftretens von ähnlicher Norm und unterschiedlicher Ergebnisrichtung bei der *Cluster-Flipping*-Methode mit Verkleinerung um den Faktor $6 \times 6 \times 1$

Obwohl für die *Cluster-Flipping*-Methode wesentlich geringere Problemanzahlen erwartet werden konnten, da sie unempfindlich gegen einzelne rechte Winkel ist, ist das Gegenteil eingetreten. Für nicht einmal jedes zweite Pixel des Ergebnisbildes kann die Richtung zweifelsfrei bestimmt werden. Etwa ein Fünftel aller Pixel hat sogar mindestens zwei zusätzliche mögliche Ergebnisrichtungen.

Betrachtet man die Lage der Pixel mit nicht eindeutiger Richtung (Abb. 4.16), dann fällt auf, dass im Gegensatz zu den bisherigen Methoden nicht nur die graue, sondern in Teilen auch die weiße Substanz betroffen ist. Zusätzlich haben sich die betroffenen Bereiche in der grauen Substanz vergrößert und verdichtet, sodass nahezu kein Vektor der grauen Substanz verlässliche Richtungsinformationen liefert. Insgesamt ist nur die Richtung jedes dritten Pixels eindeutig, weshalb auch diese Methode unbrauchbar ist.

4.4 Skalenüberbrückung durch Tensoranalyse

Keins der auf Vektoren basierenden Verfahren (*Step-by-Step*, *Flipping-the-worst* und *Cluster-Flipping*) kann eindeutige und stabile Ergebnisse liefern. Daher wurde eine weitere Verfahrensklasse überprüft, die Hauptrichtungsanalyse mit Tensoren.

4.4.1 Tensoren

Ein symmetrischer Tensor ist ein dreidimensionales Objekt, welches die Richtungsverteilung der Eingangsvektoren darstellt. Die aufspannenden Vektoren sind die Eigenvektoren einer speziellen Matrix, die aus den Datenvektoren nach folgender Vorschrift konstruiert wird:

$$T = \begin{pmatrix} T_{1,1} & T_{1,2} & T_{1,3} \\ T_{2,1} & T_{2,2} & T_{2,3} \\ T_{3,1} & T_{3,2} & T_{3,3} \end{pmatrix} = \sum_{i=0}^{Anzahl} \overrightarrow{v_i} \cdot \overrightarrow{v_i}^T = \sum_{i=0}^{Anzahl} \begin{pmatrix} v_{i_x} \cdot v_{i_x} & v_{i_x} \cdot v_{i_y} & v_{i_x} \cdot v_{i_z} \\ v_{i_y} \cdot v_{i_x} & v_{i_y} \cdot v_{i_y} & v_{i_y} \cdot v_{i_z} \\ v_{i_z} \cdot v_{i_x} & v_{i_z} \cdot v_{i_y} & v_{i_z} \cdot v_{i_z} \end{pmatrix} \quad (4.3)$$

Da die Matrix symmetrisch ist, sind ihre Eigenwerte immer reellwertig und besitzen dadurch eine Ordnung. Weiterhin gilt, dass der Eigenvektor zum größten Eigenwert die Hauptrichtung des Tensors darstellt [SCH65]. Je größer dieser Eigenwert im Verhältnis zu den beiden anderen ist, umso eindeutiger ist das Ergebnis.

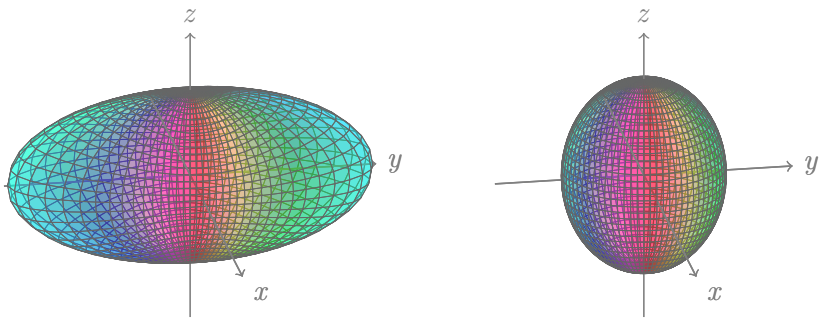


Abbildung 4.17: Beispiele für Formen eines Tensors. Der linke Tensor besitzt eine deutliche Hauptrichtung entlang der y-Achse, währed der rechte Tensor nur eine minimal stärkere Ausprägung entlang der z-Achse hat, jedoch keine klare Hauptrichtung

4.4.2 Algorithmus

Der Algorithmus selbst lässt sich in drei Teile unterteilen:

1. Bilden des Tensors aus den Vektoren nach der Vorschrift 4.3
2. Bestimmen der Eigenwerte des Tensors
3. Bestimmen der Eigenvektoren des Tensors

Zur Bestimmung der Eigenwerte des Tensors muss in einem ersten Schritt das charakteristische Polynom bzw. dessen Koeffizienten bestimmt werden.

$$
\begin{aligned}
char.\ Pol. &= \left| \begin{pmatrix} T_{1,1} - \lambda & T_{1,2} & T_{1,3} \\ T_{2,1} & T_{2,2} - \lambda & T_{2,3} \\ T_{3,1} & T_{3,2} & T_{3,3} - \lambda \end{pmatrix} \right| \\
&= a \cdot \lambda^3 \ + \ b \cdot \lambda^2 \ + \ c \cdot \lambda \ + \ d \qquad (4.4) \\
&= -1 \cdot \lambda^3 \ + \ (T_{1,1} + T_{2,2} + T_{3,3}) \cdot \lambda^2 \\
&\qquad + \ (T_{1,2}^2 + T_{1,3}^2 + T_{2,3}^2 - T_{2,2} \cdot T_{3,3} \\
&\qquad\quad - T_{1,1} \cdot T_{3,3} - T_{1,1} \cdot T_{2,2}) \cdot \lambda \\
&\qquad + \ (T_{1,1} \cdot T_{2,2} \cdot T_{3,3} + 2 \cdot T_{1,2} \cdot T_{1,3} \cdot T_{2,3} \\
&\qquad\quad - T_{1,2}^2 \cdot T_{3,3} - T_{1,3}^2 \cdot T_{2,2} - T_{2,3}^2 \cdot T_{1,1})
\end{aligned}
$$

Die Eigenwerte, d.h. die Nullstellen dieses Polynoms lassen sich zwar direkt berechnen, jedoch ist der Rechenaufwand dafür sehr hoch, da mit komplexen Zahlen gearbeitet wird. Aus diesem Grund wird nicht der direkte Weg gewählt, sondern der Weg über die Reduzierung des Polynoms zur quadratischen Gleichung. Die erste Nullstelle wird mit Hilfe des Newton-Verfahrens bestimmt. Die Iterationsvorschrift

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \tag{4.5}$$

lässt sich dabei leicht aufstellen, da es sich bei $f(x)$ um ein Polynom handelt. Einsetzen des charakteristischen Polynoms 4.4 in die Formel des Newton-Verfahrens 4.5 liefert

$$x_{n+1} = x_n - \frac{a \cdot x_n^3 + b \cdot x_n^2 + c \cdot x_n + d}{3a \cdot x_n^2 + 2b \cdot x_n + c} \tag{4.6}$$

$$N_1 = \lim_{n \to \infty} x_n \tag{4.7}$$

Die Fixpunktiteration 4.6 wird so lange wiederholt, bis sich der Wert x_n nur noch um ein gegebenes ε verändert. Dieses x_n entspricht dann der ersten Nullstelle N_1 des Polynoms und kann mit Hilfe von Polynomdivision entfernt werden.

$$\begin{array}{rrrl}
(a\lambda^3+ & b\lambda^2 + & c\lambda + d): (\lambda - N_1) = a\lambda^2 + (b + aN_1)\lambda & (4.8)\\
-(a\lambda^3- & aN_1\lambda^2) & + c + bN_1 + aN_1^2 & \\
\hline
 & (b + aN_1)\lambda^2 + & c\lambda & \\
 & -((b + aN_1)\lambda^2 - & (bN_1 + aN_1^2)\lambda) & \\
\hline
 & & (c + bN_1 + aN_1^2)\lambda + d & \\
 & & -((c + bN_1 + aN_1^2)\lambda + d) & \\
 & & \hline
 & & 0 &
\end{array}$$

Anmerkung: Im letzten Schritt muss sich zwangsläufig die 0 ergeben, da N_1 Nullstelle des Polynoms ist.

Zur Bestimmung des zweiten und dritten Eigenwertes der Matrix gilt es also, die Nullstellen der quadratischen Gleichung 4.8 zu finden. Division durch a und Anwenden der pq-Formel liefert:

$$N_2 = -\frac{b + aN_1}{2a} + \sqrt{\frac{(b + aN_1)^2}{4a^2} - \frac{(c + bN_1 + aN_1^2)}{a}} \tag{4.9}$$

$$N_3 = -\frac{b + aN_1}{2a} - \sqrt{\frac{(b + aN_1)^2}{4a^2} - \frac{(c + bN_1 + aN_1^2)}{a}} \tag{4.10}$$

Um im dritten Abschnitt des Algorithmus die Eigenvektoren zu bestimmen, muss für jeden Eigenwert, bzw. für jede Nullstelle N_i des charakteristischen Polynoms der Gauß-Algorithmus auf folgende Matrix angewendet werden:

$$\begin{pmatrix} T_{1,1} - N_i & T_{1,2} & T_{1,3} \\ T_{2,1} & T_{2,2} - N_i & T_{2,3} \\ T_{3,1} & T_{3,2} & T_{3,3} - N_i \end{pmatrix} \tag{4.11}$$

Da sich in der letzte Zeile immer eine Nullzeile ergibt, wird der Komponente x_3 jedes Eigenvektors $\overrightarrow{x_i}$ eine 1 zugewiesen. Rückwärtseinsetzen liefert dann x_2 und x_1. Um den PLI-Eigenschaften gerecht zu werden, normiert man die Eigenvektoren. Ergebnisrichtung ist der Eigenvektor zum betragsgrößten Eigenwert.

Tensor **erstelleTensor**(Vektor[] `feld`)

Initialisiere Tensor T mit Nullen
Für alle Indices `i`=0 des Vektorfeldes
$T_{1,1} = T_{1,1}$ + `feld[i].x` * `feld[i].x`
$T_{1,2} = T_{1,2}$ + `feld[i].x` * `feld[i].y`
$T_{1,3} = T_{1,3}$ + `feld[i].x` * `feld[i].z`
$T_{2,1} = T_{2,1}$ + `feld[i].y` * `feld[i].x`
$T_{2,2} = T_{2,2}$ + `feld[i].y` * `feld[i].y`
$T_{2,3} = T_{2,3}$ + `feld[i].y` * `feld[i].z`
$T_{3,1} = T_{3,1}$ + `feld[i].z` * `feld[i].x`
$T_{3,2} = T_{3,2}$ + `feld[i].z` * `feld[i].y`
$T_{3,3} = T_{3,3}$ + `feld[i].z` * `feld[i].z`
T ist jetzt der fertige Tensor

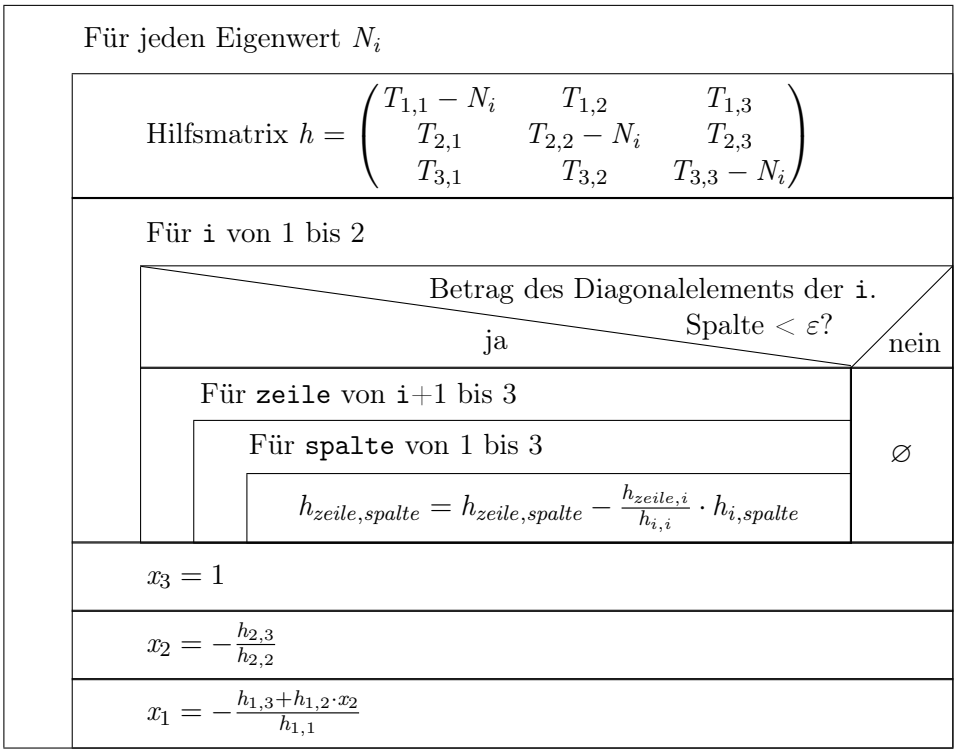
Nassi 4.5: Erstellen des Tensors

bestimmeEigenwerte(Tensor T)

$a = -1$
$b = T_{1,1} + T_{2,2} + T_{3,3}$
$c = T_{1,2}^2 + T_{1,3}^2 + T_{2,3}^2 - T_{2,2} \cdot T_{3,3} - T_{1,1} \cdot T_{3,3} - T_{1,1} \cdot T_{2,2}$
$d = T_{1,1} \cdot T_{2,2} \cdot T_{3,3} + 2 \cdot T_{1,2} \cdot T_{1,3} \cdot T_{2,3} - T_{1,2}^2 \cdot T_{3,3} - T_{1,3}^2 \cdot T_{2,2} - T_{2,3}^2 \cdot T_{1,1}$
$N_1 = 0$
Solange $\left\lvert aN_1^3 + bN_1^2 + cN_1 + d \right\rvert > \varepsilon$
$N_1 = N_1 - \frac{aN_1^3+bN_1^2+cN_1+d}{3aN_1^2+2bN_1+c}$
$N_2 = -\frac{b+aN_1}{2a} + \sqrt{\frac{(b+aN_1)^2}{4a^2} - \frac{(c+bN_1+aN_1^2)}{a}}$
$N_3 = -\frac{b+aN_1}{2a} - \sqrt{\frac{(b+aN_1)^2}{4a^2} - \frac{(c+bN_1+aN_1^2)}{a}}$

Nassi 4.6: Bestimmen der Eigenwerte des Tensors

bestimmeEigenvektoren(Tensor T, double N_1, double N_2, double N_3)

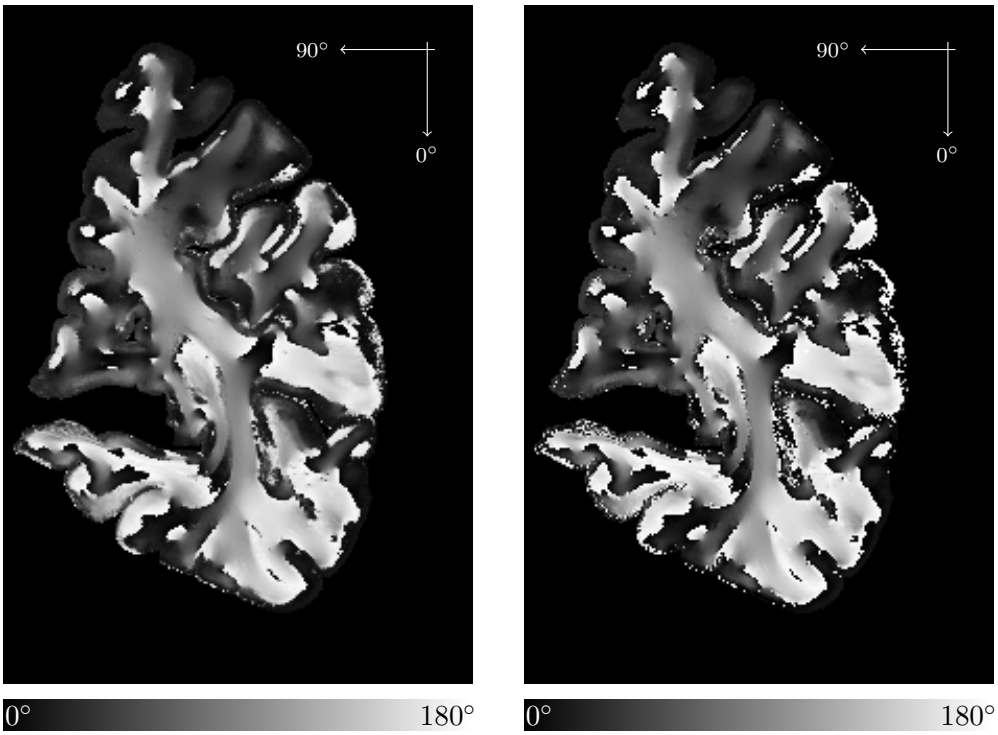


Nassi 4.7: Bestimmen der Eigenvektoren des Tensors

4.4.3 Auswertung

Obwohl das Ergebnisbild der Tensoranalyse (Abb. 4.18(b)) sehr ähnlich zum Ergebnisbild der *Flipping-the-worst*-Methode (Abb. 4.10(b) auf Seite 21) ist, steigt die Qualität der Ergebnisbilder stark an. Es treten keine Probleme mehr bei rechten Winkeln oder ähnlichen Skalarprodukten auf. Ein Problem existiert bei der Tensoranalyse dann, wenn der betragsgrößte Eigenwert nicht wesentlich größer als der Betrag des zweit- und/oder drittgrößten Eigenwertes ist, während sich die Vektoren merklich voneinander unterscheiden (Abb. 4.17). Das bedeutet, dass es in diesem Fall keine klare Hauptrichtung des Tensors gibt. Um nur wirklich klare Ergebnisse zuzulassen, wird es als Problem angesehen, wenn der betragsgrößte Eigenwert nicht mindestens 50% größer als die anderen beiden ist, während der Winkel zwischen den betroffenen Eigenvektoren größer als 5 Grad ist.

Der Anteil der Problemfälle ist stark gesunken, trotz der Einschränkung, dass der betragsgrößte Eigenwert mindestens 50% größer sein muss als der Betrag der anderen beiden Eigenwerte. Nur etwa jedes zwanzigste Pixel hat eine zweite mögliche Ergebnisrichtung. Das Auftreten nahezu kugelförmiger Tensoren, d.h. dreier annähernd identischer Eigenwerte kann vernachlässigt werden (Tabelle 4.4).



(a) einfache Mittelrichtung

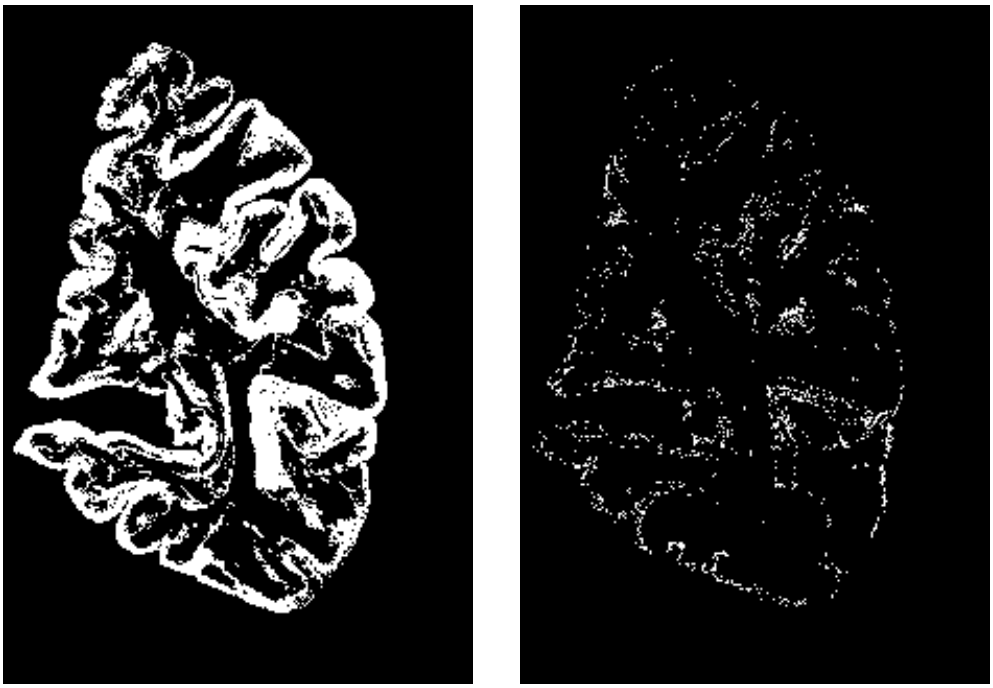
(b) Tensoranalyse

Abbildung 4.18: Ergebnisbilder (Direktion) der einfachen Mittelrichtung und der Tensoranalyse bei Verkleinerung des Originalbildes um den Faktor $6 \times 6 \times 1$.

Anzahl ähnlicher Eigenwerte	absolute Häufigkeit	relative Häufigkeit
0	4981201	94,519%
1	288779	5,480%
2	60	0,001%

Tabelle 4.4: Häufigkeit des Auftretens von ähnlichen Eigenwerten und unterschiedlicher Ergebnisrichtung bei Tensoranalyse mit Verkleinerung um den Faktor $6 \times 6 \times 1$ bei einem Mindestunterschied der Eigenwerte von 50%

Bei der Betrachtung der Lage der Pixel mit nicht eindeutiger Richtung fällt auf, dass diese in etwa der Hirnrinde entsprechen (Abb. 4.19(b)). Dort sind aufgrund geringerer Faserdichte und schwächerer Myelinisierung Messungen grundsätzlich ungenauer und können häufiger keine eindeutige Richtung liefern. Aus diesem Grund sind die bei der Tensoranalyse auftretenden Probleme eher der Messung als dem Analyseverfahren geschuldet.



(a) Abweichung zur Mittelrichtung $> 5°$ (b) Pixel mit nicht eindeutiger Richtung

Abbildung 4.19: Auswertung des Unterschiedes von einfacher Mittelrichtung und Tensoranalyse durch weiße Markierung aller Pixel, deren Richtungsdifferenz größer als 5 Grad ist. Betrachtet wird die Summe der Differenzen in Direktion und Inklination. Dem gegenüber sind alle Pixel mit mehrdeutiger Richtung, d.h. ähnlichen Eigenwerten und abweichender Ergebnisrichtung, ebenfalls weiß markiert.

Ein Vergleich mit dem Differenzbild zu den aus Mittelrichtungen bestimmten Vektoren zeigt außerdem, dass von den vielen Pixeln, die vom einfachen Mittelwert abweichen, nur ein kleiner Teil eine zweite mögliche Richtung aufweist (Abb. 4.19(a)). Das bedeutet, dass diese Methode trotz des strengen Kriteriums zuverlässige Ergebnisse liefert. Nimmt man ein weniger strenges Kriterium, wie etwa eine Schwelle der Eigenwerte von 25% Differenz anstatt 50%, so sinkt der Anteil der Pixel mit mehreren möglichen Ergebnisrichtungen bereits auf deutlich unter 2% ab (Tab. 4.5).

Anzahl ähnlicher Eigenwerte	absolute Häufigkeit	relative Häufigkeit
0	5180013	98,292%
1	90025	1,708%
2	2	0,000%

Tabelle 4.5: Häufigkeit des Auftretens von ähnlichen Eigenwerten und unterschiedlicher Ergebnisrichtung bei Tensoranalyse mit Verkleinerung um den Faktor $6 \times 6 \times 1$ bei einem Mindestunterschied der Eigenwerte von 25%

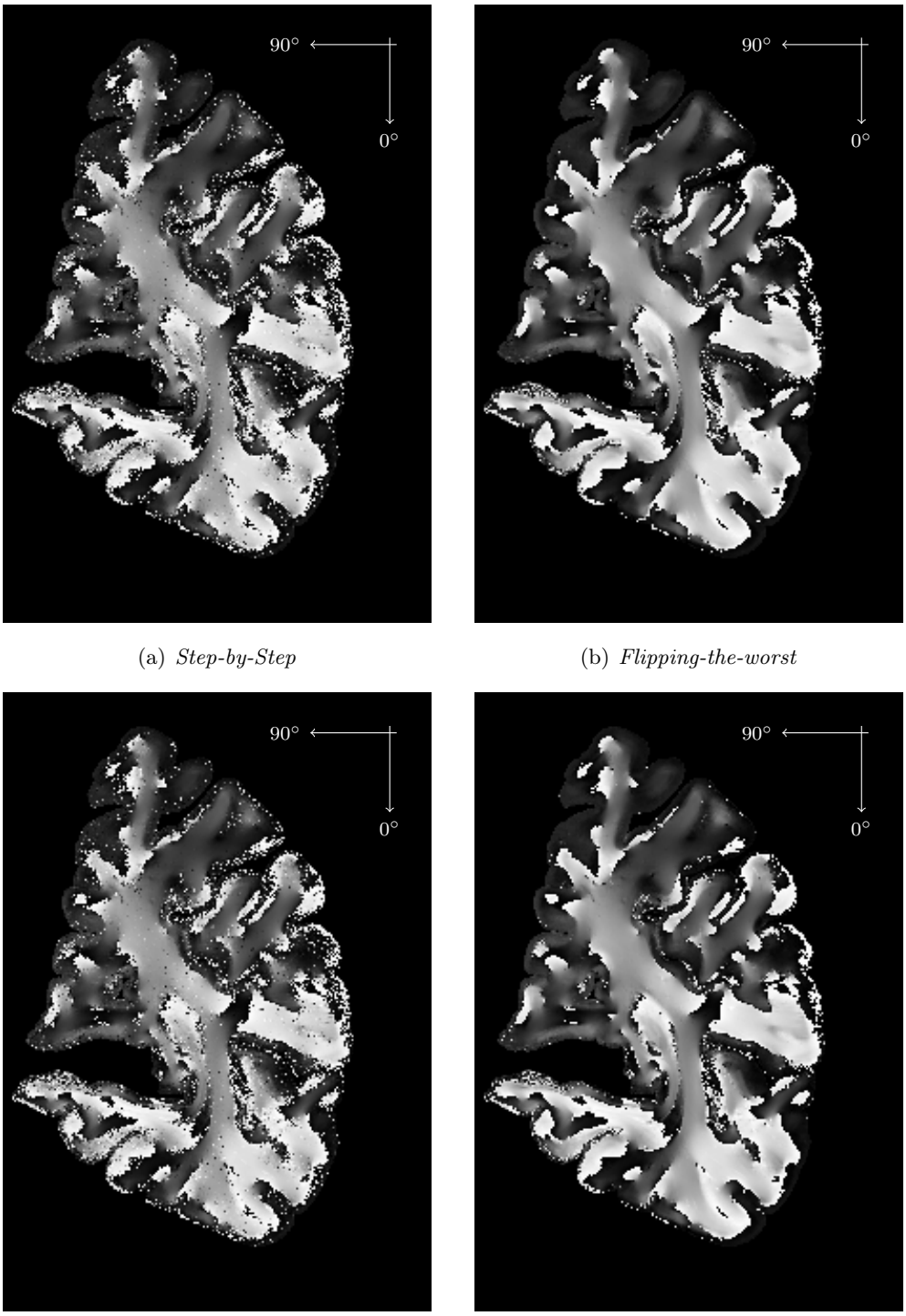
4.5 Vergleich der Ergebnisse

Beim Vergleich der Ergebnisbilder untereinander (siehe Abb. 4.20) fällt auf, dass die Methoden *Step-by-Step* und *Cluster-Flipping* stark verrauschte Ergebnisse liefern, während die Ergebnisbilder der *Flipping-the-worst*-Methode und der Tensoranalyse wesentlich glatter sind. Diese entsprechen dadurch eher dem Originalbild als die beiden anderen Bilder.

Ähnliches zeigen auch die Plots der Ergebnisrichtungen. Während *Step-by-Step* und vor allem *Cluster-Flipping* stark inhomogene Ergebnisse liefern, sind die Bilder, die durch *Flipping-the-worst* und insbesondere die Tensoranalyse erzeugt wurden, sehr homogen und passen sehr gut zum Originalbild. Ein Ausschnitt der Ergebnisrichtungen ist zu sehen in Abbildung 4.21.

Die Farbe des Richtungspfeils setzt sich dabei aus seinen Richtungskomponenten zusammen. Rot steht für die x-Komponente (horizontal), grün für die y-Komponente (vertikal) und blau für die z-Komponente (senkrecht zur Schnittebene).

Alle im Verlauf dieses Kapitels genannten Ergebnisse führen dazu, dass ausschließlich die Tensoranalyse als zuverlässiges und korrekt arbeitendes Verfahren genutzt werden sollte.



(a) *Step-by-Step*

(b) *Flipping-the-worst*

(c) *Clusterflipping*

(d) Tensoranalyse

Abbildung 4.20: Ergebnisbilder der Direktion bei Verarbeitung des Bildes mit den verschiedenen Methoden. Ein weißes Pixel steht für einen Winkel von 180°, ein schwarzes für einen Winkel von 0°

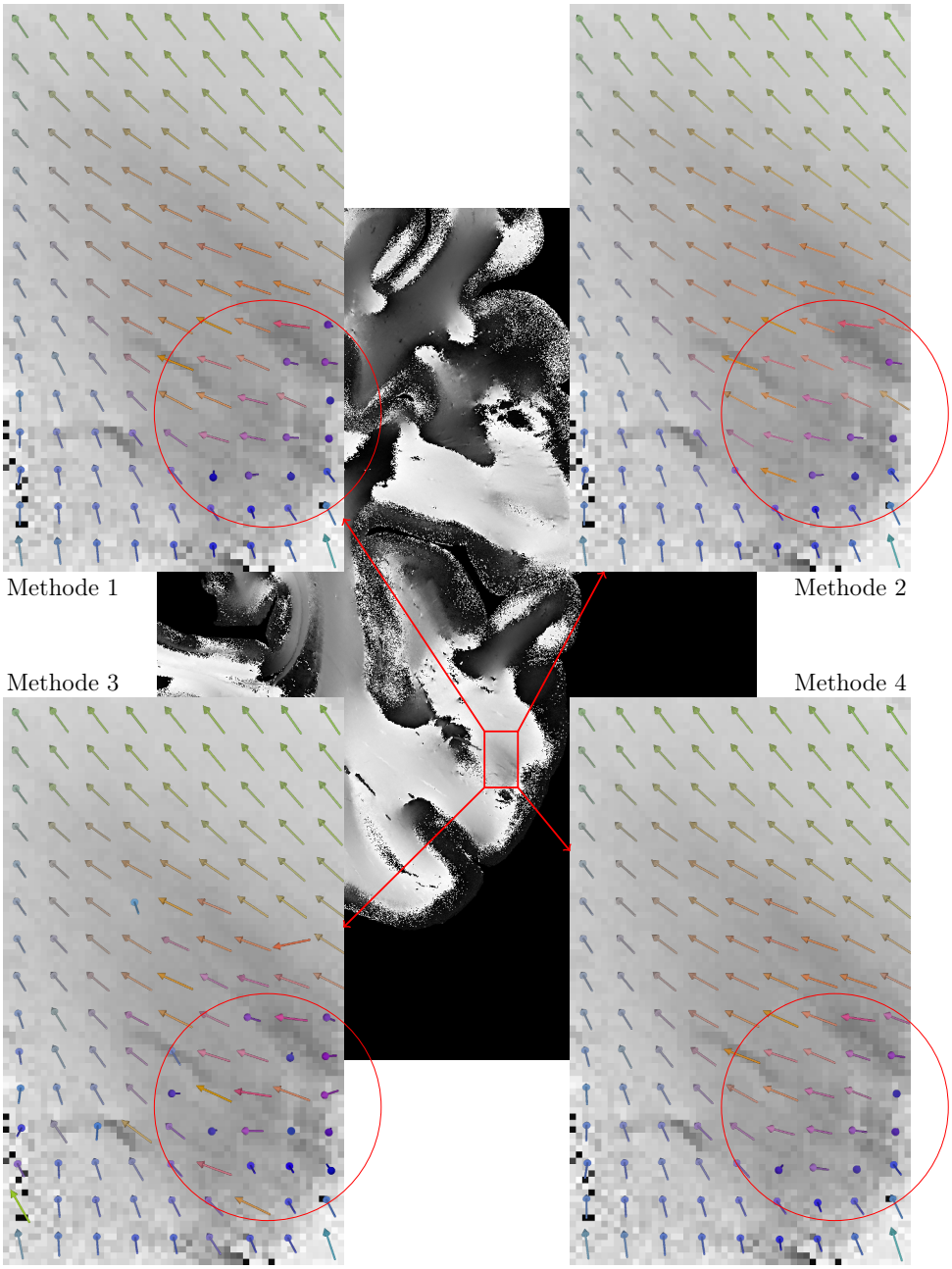


Abbildung 4.21: Verschiedene Ergebnisrichtungen bei Verarbeitung des Bildes mit Methode 1 (sequentielle Verarbeitung mit Invertierung bei Winkeln $> 90°$), Methode 2 (Invertierung von Vektoren bis alle Winkel zur Mittelrichtung spitz sind), Methode 3 (Cluster-Flipping) und Tensoranalyse (Methode 4, Hauptrichtung ist Eigenvektor zum größten Eigenwert). Ein schwarzes Pixel steht für einen Direktionswinkel von 0 Grad und ein weißes Pixel für einen Winkel von 180 Grad. Die Nullrichtung verläuft dabei senkrecht von oben nach unten.

5 Parallelisierung

Die Implementierung wurde mit einem Datensatz von 160 Schnitten mit einer Größe von 1950×1350 Pixeln getestet. Die Verarbeitung benötigte eine Laufzeit von 40 Sekunden bei einer Verkleinerung um den Faktor $6 \times 6 \times 1$. Bei linearem Laufzeitverhalten im Verhältnis zur Größe würde das bedeuten, dass die Verarbeitung eines Gesamthirns von 2000 Schnitten mit einer Größe von 50000×30000 Pixeln mehr als drei Tage dauern würde. Um die Laufzeit zu reduzieren, sollte das Programm parallelisiert werden.

5.1 Parellelisierungsstrategien

Bei der Parallelisierung eines Programmes gibt es verschiedene mögliche Strategien, die Arbeit zwischen allen beteiligten Prozessoren aufzuteilen. Die Aufteilung der Arbeit geschieht in diesem Anwendungsfall über die Verteilung der zu berechnenden Schnitte auf die Prozessoren. Diese können unabhängig voneinander berechnet werden, wodurch keine Kommunikation zwischen den Prozessoren nötig ist.

Die erste Verteilungsstrategie, die **blockweise Verteilung**, weist jedem Prozessor mit Index r von insgesamt p Prozessoren einen zusammenhängenden Block der insgesamt s Schnitte zu. Die Nummern des ersten und letzten zu verarbeitenden Schnittes eines Prozessores lassen sich dabei wie folgt berechnen:

$$x_{Start} = \lfloor \frac{r \cdot s}{p} \rfloor \tag{5.1}$$

$$x_{Ende} = \lfloor \frac{(r+1) \cdot s}{p} - 1 \rfloor \tag{5.2}$$

Prozessornummer	x_{Start}	x_{Ende}	Anzahl
0	$\lfloor \frac{0 \cdot 7}{4} \rfloor = 0$	$\lfloor \frac{1 \cdot 7}{4} - 1 \rfloor = 0$	1
1	$\lfloor \frac{1 \cdot 7}{4} \rfloor = 1$	$\lfloor \frac{2 \cdot 7}{4} - 1 \rfloor = 2$	2
2	$\lfloor \frac{2 \cdot 7}{4} \rfloor = 3$	$\lfloor \frac{3 \cdot 7}{4} - 1 \rfloor = 4$	2
3	$\lfloor \frac{3 \cdot 7}{4} \rfloor = 5$	$\lfloor \frac{4 \cdot 7}{4} - 1 \rfloor = 6$	2

Tabelle 5.1: Verteilung von 7 Schnitten (nummeriert von 0 bis 6) auf 4 Prozessoren mit blockweiser Verteilung

Die zweite Möglichkeit ist die **zyklische Verteilung**. Dabei erhält jeder Prozessor der Reihe nach einen zu berechnenden Schnitt. Die Zuteilung erfolgt über Schnittnummer modulo Prozessoranzahl. Sie ist ebenso wie die blockweise Verteilung eine statische Strategie, da Zuteilungen bereits vorher bekannt sind.

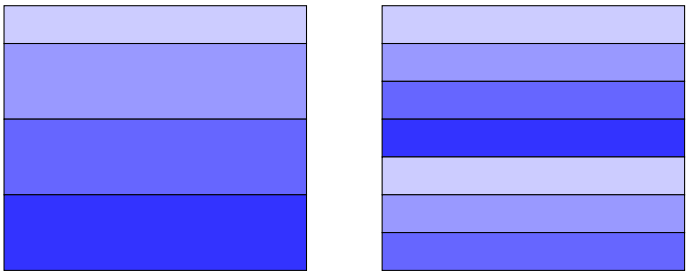


Abbildung 5.1: Blockweise (links) und zyklische (rechts) Verteilung von 7 Schnitten an 4 Prozessoren

Eine weitere Strategie ist das **Master/Worker-Schema**. Dabei übernimmt ein Prozessor die Arbeit des Masters, alle anderen Prozessoren fungieren als Worker. Der Master nimmt dabei dynamische Lastverteilung vor, indem er den Workern immer dann neue Arbeit zuteilt, wenn sie ihre bisherige Arbeit beendet haben. Der Master-Prozessor selbst übernimmt dabei keine Berechnungen, sondern nur die Zuteilung. Diese Lastverteilung heißt dynamisch, weil nicht vorhergesagt werden kann, welcher Prozessor welchen Schnitt bearbeiten wird.

5.2 Laufzeitverhalten

Bei der Überprüfung der Laufzeiten muss sichergestellt sein, dass alle Testläufe mit identischen Parametern stattfinden. Wie bei den bereits gezeigten Ergebnisbildern ist der gewählte Verkleinerungsfaktor $6 \times 6 \times 1$. Für alle drei Parallelisierungsstrategien wird das Programm mit 1, 2, 4, 8 und 16 Prozessoren ausgeführt und die Laufzeit gemessen.

Die gewählte Interpolationsmethode, für die die Strategien verglichen werden, ist die *Cluster-Flipping*-Methode. Trotz der Tatsache, dass sie nicht die besten Ergebnisse liefert, wurde sie für die Laufzeitmessungen ausgewählt, da für die Tensormethode wegen des zu kleinen Datensatzes keine belastbaren Messungen durchgeführt werden konnten.

Betrachtet man das Diagramm der Laufzeiten (Abb. 5.2), fällt auf, dass zyklische Verteilung und blockweise Verteilung nahezu identische Laufzeiten aufweisen. Das Master/Worker-Schema ist bei Ausführung mit bis zu 4 Kernen langsamer, da einer der Kerne keine Berechnungen vornimmt, sondern nur die zu bearbeitenden Schnitte auf die anderen Kerne verteilt. Das bedeutet, dass bei einem Programmaufruf mit n Kernen effektiv nur n-1 Kerne rechnen. Trotzdem sind bei einem Aufruf mit 8 Kernen alle Methoden gleich schnell. Bei Aufruf mit 16 Kernen ist das Master/Worker-Schema am schnellsten.

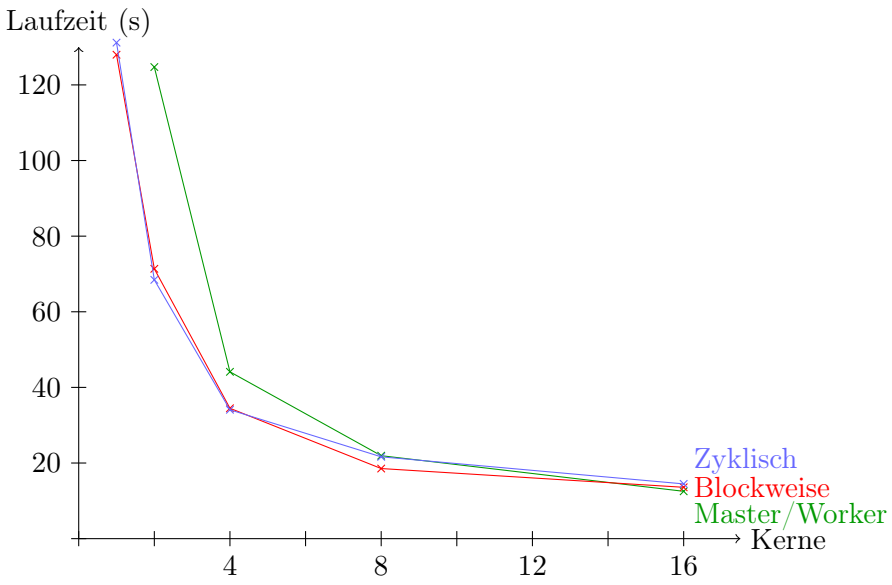


Abbildung 5.2: Laufzeiten der einzelnen Strategien bei Aufruf mit verschiedenen Kernanzahlen und Vekleinerungsfaktor $6 \times 6 \times 1$

Um zu verstehen, warum diese Methode schneller ist, obwohl ein Kern weniger rechnet, muss man die Auslastung der einzelnen Prozessorkerne betrachten. Dafür wird die Zeit gemessen, die ein einzelner Kern aktiv ist, und in Relation zur Gesamtrechenzeit des Programmes, d.h. des langsamsten Kerns gesetzt. Überprüft wurde dabei die Messung der Strategien mit 16 Kernen.

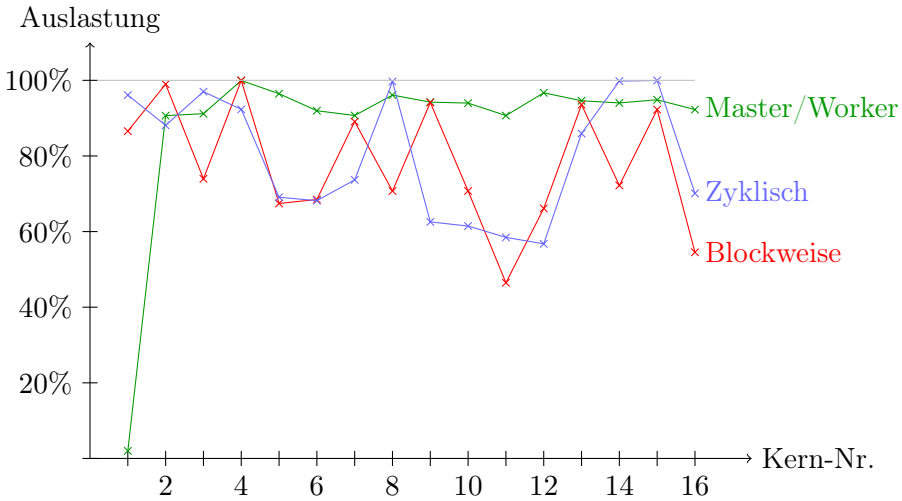


Abbildung 5.3: Auslastung der Kerne bei Aufruf des Programmes mit 16 Kernen für die verschiedenen Parallelisierungsstrategien

In Abb. 5.3 sieht man, dass die Auslastung der rechnenden Kerne (2-16) bei der Master/Worker-Methode immer mindestens 90% beträgt, während bei den anderen Methoden einzelne Kerne zum Teil mehr als ein Drittel der Zeit inaktiv sind. Die durchschnittliche Auslastung der Kerne liegt sowohl bei blockweiser, als auch bei zyklischer Verteilung unter 80%, weshalb die Gesamtlaufzeit bei diesen Methoden größer ist. Die schlechtere Auslastung der Kerne liegt im unterschiedlichen Rechenaufwand pro Schnitt und dem Erreichen von Gesamt-I/O-Limits begründet, weshalb einzelne Kerne manchmal länger zum Einlesen bzw. Schreiben von Daten brauchen.

Um diesen Vorteil des Master/Worker-Schemas zu bestätigen, kann man die Laufzeiten für verschiedene Verkleinerungsstufen miteinander vergleichen. Dafür wurde das Programm mit 16 Kernen und den Verkleinerungsstufen 2×2, 4×4, 6×6, 8×8 und 10×10 aufgerufen, jeweils ohne schnittübergreifende Verkleinerung.

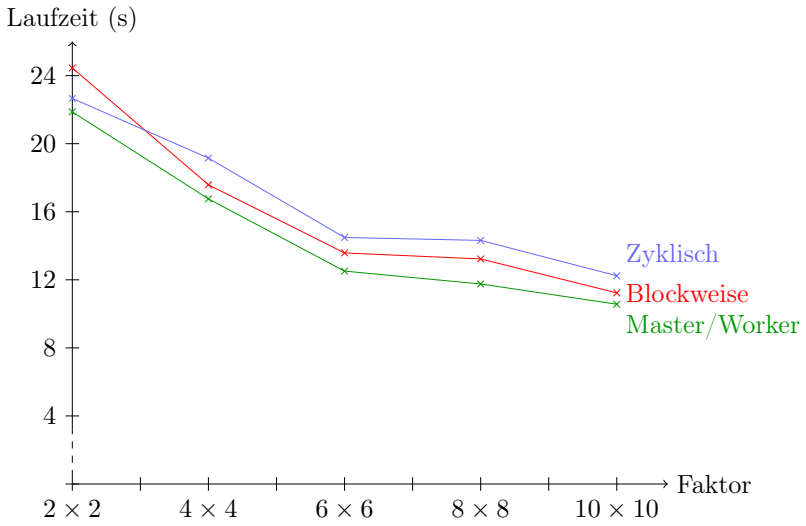


Abbildung 5.4: Laufzeiten der einzelnen Strategien bei 16 Kernen und verschiedenen Vekleinerungsfaktoren

Diese Laufzeitmessungen zeigen, dass das Master/Worker-Schema schneller ist als die beiden anderen Strategien.

Es fällt auf, dass die Laufzeit mit steigendem Verkleinerungsfaktor sinkt. Der Grund hierfür ist die Größe der Ausgabedatei. Diese wird kleiner, wenn der Verkleinerungsfaktor erhöht wird. Dadurch sinkt auch der Schreibaufwand.

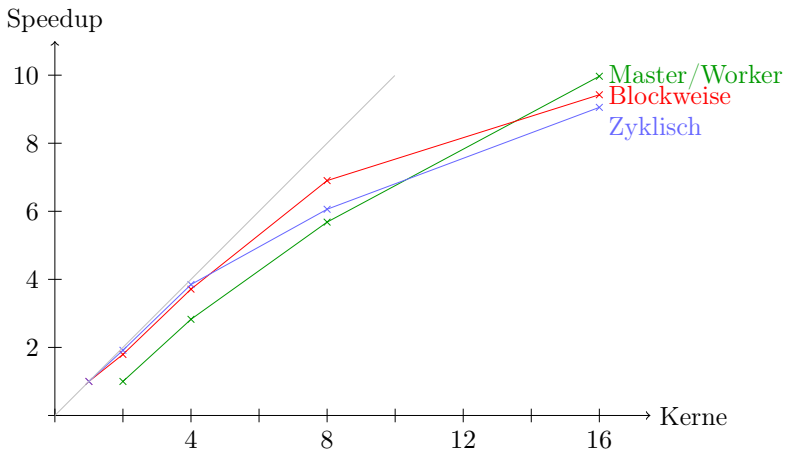


Abbildung 5.5: Speedup der einzelnen Strategien

Um das Laufzeitverhalten abschließend beurteilen zu können, muss der Speedup-Graph aus den Laufzeiten gebildet und ausgewertet werden. Auch dieser zeigt, dass das Master/Worker-Schema bei größeren Prozessorzahlen besser ist. Ebenso zeigt er, dass dieses Schema besser skaliert als die beiden anderen, da die Kurve steiler verläuft.

In Abb. 5.5 ist zu sehen, dass bereits mit 16 Kernen im Master/Worker-Schema ein Speedup von 10 erreicht werden kann, wodurch die Bearbeitungszeit eines gesamten Gehirnes von über drei Tagen auf etwa acht Stunden sinkt. Diese Bearbeitungszeit lässt sich noch weiter reduzieren durch den Einsatz zusätzlicher Prozessorkerne. Da der Beispieldatensatz aber zu klein war, um mit mehr als 16 Prozessoren belastbare Laufzeitmessungen vorzunehmen, konnte das nicht überprüft werden.

6 Zusammenfassung und Ausblick

In dieser Arbeit wurden Methoden zur Vereinigung von Richtungsinformationen aus PLI-Bildern erarbeitet. PLI-Bilder sind durch die Technik des Polarized Light Imaging (PLI) aufgenommene Bilder von Gehirnschnitten. Mit Hilfe des PLI-Rekonstruktionsworkflows können aus ihnen Informationen über die räumliche Orientierung der Fasern gewonnen werden (Kapitel 2).

Dabei werden unter anderem der Winkel einer Faser in der Schnittebene und der Winkel zwischen einer Faser und ihrer Projektion auf die Schnittebene berechnet. Zusätzlich ist es möglich, aus den Rohdaten eine Maske zur Trennung von Hintergrund und grauer bzw. weißer Substanz des Gehirnes zu bestimmen.

Diese Daten werden im speziell für große Datenmenge entwickelten Format HDF5 gespeichert (Kapitel 3). Sie enthalten alle nötigen Informationen, um ein Gesamtvolumenbild zu rekonstruieren.

Die Ergebnisse dieser Rekonstruktion und somit auch der Workflow an sich können bisher nicht verifiziert werden. Dafür muss man die Ergebnisse mit bekannten korrekten Daten vergleichen. Eine Möglichkeit dabei sind MRT-Bilder, die jedoch eine wesentlich geringere Auflösung bieten. Um die vorhandenen PLI-Bilder auf MRT-Auflösung reduzieren zu können, wurden in Kapitel 4 mehrere verschiedene Ansätze überprüft.

Dabei sind alle drei überprüften Methoden, die auf Vektorinterpolation basieren, an verschiedenen Problemen gescheitert. Grund hierfür ist die Tatsache, dass PLI-Bilder nur Orientierungen anstelle von Richtungen darstellen können. Das bedeutet, dass in PLI-Bildern die Vektoren (1, 1, 1) und (-1, -1, -1) nicht unterscheidbar sind, weshalb es schwierig ist, eine Hauptrichtung von mehreren Orientierungen zu finden. Die Eindeutigkeit der Ergebnisse der vektorbasierten Methoden konnte nicht garantiert werden.

Gelöst werden konnten diese Probleme durch die Anwendung der Tensoranalyse. Diese erzeugt aus den zu verarbeitenden Vektoren eine Matrix, deren Eigenvektoren einen Ellipsoid aufspannen. Die Hauptrichtung des Ellipsoids, d.h. der Eigenvektor zum größten Eigenwert, ist dabei die Hauptrichtung der verarbeiteten Vektoren.

Durch Parallelisierung der Implementierung konnte die Laufzeit erheblich gesenkt werden. Dafür wurden verschiedene Parallelisierungsstrategien miteinander verglichen (Kapitel 5).

Für eine zukünftige Version ist denkbar, die Auswertung der Tensoren, d.h. das Finden der Eigenwerte und den zugehörigen Vektoren auf GPUs auszulagern, da die einzelnen Tensoren vollkommen unabhängig voneinander verarbeitbar sind.

Auch eine Optimierung der Lastverteilung beim Master/Worker-Schema ist möglich, sodass nicht mehr nur einzelne Schnitte verteilt werden, sondern beispielsweise immer kleiner werdende Pakete von Schnitten, weil so der Kommunikationsaufwand reduziert werden kann.

Supercomputer

Juelich Dedicated GPU Environment (JuDGE)

Der Supercomputer JuDGE steht allen Mitarbeitern im Juelich Supercomputing Centre für Simulationen zur Verfügung [JSC14].

Rechenknoten

- 206 Rechenknoten IBM System x iDataPlex dx360 M3
  - Compute Node: 2 Intel Xeon X5650(Westmere) 6-Kern Prozessoren (2,66 GHz)
  - Hauptspeicher: 96 GB
  - GPU: 2 NVIDIA Tesla M2050 (Fermi) 1,15 GHz(448 Kerne), 3 GB Hauptspeicher (auf 54 Knoten)
  - GPU: 2 NVIDIA Tesla M2070 (Fermi) 1,15 GHz(448 Kerne), 6 GB Hauptspeicher (auf 152 Knoten)

Gesamtsystem

- 2472 Kerne insgesamt
- 412 Grafikprozessoren
- 19,8 TB Hauptspeicher
- 239 Teraflops Peak Performance

Infrastrukturknoten

- GPFS (Dateisystem) Verbindungsknoten
  - 2 IBM System x iDataPlex dx360 M3 (8 Kerne, 12 GB Hauptspeicher)
- Loginknoten und Verwaltungsknoten
  - 4 IBM System x iDataPlex dx360 M3 (12 Kerne, 96 GB Hauptspeicher)

Literaturverzeichnis

[BMBF] http://www.bmbf.de/press/3413.php, 2013. Letzter Zugriff: 03.08.2015

[JSC14] http://www.fz-juelich.de/ias/jsc/EN/Expertise/Supercomputers/JUDGE/Configuration/Configuration_node.html, 2014. Letzter Zugriff: 03.08.2015

[AA+11] Markus Axer, Katrin Amunts, David Grässel, Christoph Palm, Jürgen Dammers, Hubertus Axer, Uwe Pietrzyk, Karl Zilles – A novel approach to the human connectome: Ultra-high resolution mapping of fiber tracts in the brain, *Neuroimage*, 54:1091 - 1101, 2011.

[AK14] https://andrewlawrenceking.files.wordpress.com/2014/10/myelin.jpg, 2014. Letzter Zugriff: 03.08.2015

[SCH65] A.E. Scheidegger, Urbana, Ill. – On the Statistics of the orientation of bedding planes, grain axes, and similar sedimentological data, 1965

[HDF5] https://www.hdfgroup.org/HDF-FAQ.html#6b, 11.02.2015. Letzter Zugriff: 03.08.2015

[NASA] http://newsroom.gsfc.nasa.gov/sdptoolkit/hdfeosfaq.html#General, 2009. Letzter Zugriff: 03.08.2015

[AC+09] http://www.ncbi.nlm.nih.gov/pubmed/19226510, 2009. Letzter Zugriff: 03.08.2015

[SHL11] Robert F. Schmidt, Manfred Heckmann, Florian Lang – Physiologie des Menschen: mit Pathophysiologie, S. 72, 31. Auflage, Springer-Verlag, DOI: http://dx.doi.org/10.1007/978-3-642-01651-6