Ruprecht-Karls-Universität Heidelberg

Hochschule Heilbronn

Diplomstudiengang Medizinische Informatik
der Universität Heidelberg/Hochschule Heilbronn

Diplomarbeit

Robotergestützte Ultraschalltomographie

Tim Beyl
t.beyl@gmx.de

09.06.2010-09.12.2010

Erstgutachter: Prof. Dr. Oliver Kalthoff
Zweitgutachter: Prof. Dr. Heinz Wörn
Betreuer: Dipl.-Inform. Thorsten Brennecke

Erklärung

Hiermit erkläre ich, dass ich die vorliegende Diplomarbeit selbständig angefertigt habe. Es wurden nur die in der Arbeit ausdrücklich benannten Quellen und Hilfsmittel benutzt. Wörtlich oder sinngemäß übernommenes Gedankengut habe ich als solches kenntlich gemacht.

Ort, Datum	Unterschrift

Danksagung

Ich bedanke mich bei Herrn Prof. Dr. Kalthoff für die Betreuung der Diplomarbeit. Insbesondere seine Empfehlungen bezüglich 3D Slicer konnten gewinnbringend umgesetzt werden.

Des Weiteren bedanke ich mich bei Herrn Dr. Raczkowsky für die Zustimmung zu dieser Diplomarbeit in seiner Arbeitsgruppe (MeGI) am IPR.

Frau Dr. Jessica Burgner (KIT) danke ich für die Betreuung der Diplomarbeit in der Einarbeitungsphase und für die Einführung in die am Institut für Prozessrechentechnik, Automation und Robotik verwendeten Software- und Hardwareprodukte.

Besonderer Dank geht an Herrn Thorsten Brennecke (KIT), der über den kompletten Bearbeitungszeitraum die Betreuung für die Diplomarbeit übernommen und sich immer Zeit für fachliche und inhaltliche Fragen genommen hat.

Weiterer Dank geht an Herrn Junichi Tokuda (Harvard University), der mich bezüglich Fragen zur Entwicklung eigener Erweiterungen für OpenIGTLink und OpenIGTLinkIF per e-mail immer zeitnah unterstützte.

Herrn Holger Mönnich(KIT) möchte ich für die freundliche Bereitstellung der Quellen zur Ansteuerung des KUKA LWR IV aus Matlab danken.

Herrn Daniel Stein (KIT) danke ich für die Unterstützung bezüglich mathematischer Fragen, die sich während des Verlaufs der Arbeit ergaben.

Herrn Hartmut Regner möchte ich dafür danken dass er jegliches mechanische Problem, das während der Arbeit auftrat, mit größtem Engagement zeitnah löste.

Besonderer Dank geht an meine Eltern, die mich während des kompletten Studiums unterstützten und jederzeit hinter mir standen.

Zusammenfassung

Ultraschallbildgebung ist ein in der Medizin häufig verwendetes Verfahren zur Diagnostik und Verlaufskontrolle. Ultraschall bietet vor allem eine große Weichteilauflösung bei gleichzeitig niedrigen Kosten, erfordert aber vom Anwender ein großes Maß an Erfahrung. Aufgrund vieler Artefakte und Verzerrungen in Ultraschallbildern ist die Bildqualität der Bilder deutlich geringer, als die derer, die mittels Magnetresonanztomographie erstellt werden können. Deshalb arbeiten viele verschiedene Forschungsgruppen an der Verbesserung der Modalitäten und Datensätze im Zusammenhang mit Ultraschallbildgebung.

Diese Arbeit beschäftigt sich mit der Entwicklung eines robotergestützten Ultraschalltomographen. Das Gerät ist in der Lage tomographische Bildserien von Phantomen und Weichgewebe zu erstellen, die anschließend dreidimensional visualisiert werden können.

Dabei wird eine möglichst große Bildqualität und Ortsauflösung angestrebt. Der Ultraschalltomograph soll als Entwicklungsplattform für weitere Entwicklungen im Bereich des dreidimensionalen Ultraschall dienen.

Die Arbeit basiert hierbei auf mehreren verschiedenen Robotersystemen und dem DiPhAS Ultraschallsystem des Fraunhoferinstituts für biomedizinische Technik und wurde in der Medizingruppe (MeGI) des Instituts für Prozessrechentechnik, Automation und Robotik (KIT) durchgeführt.

Es werden die für die Ultraschalltomographie notwendigen Grundlagen, sowie der Stand der Forschung beleuchtet. Außerdem werden Konzept und Realisierung des Ultraschalltomographen im Detail vorgestellt. Anschließend beschäftigt sich diese Arbeit mit der Akquisition von Datensätzen vielfältiger Phantome und Unterarme mehrerer Probanden. Die Daten werden geeignet aufbereitet und visualisiert.

Schließlich wird das entwickelte System hinsichtlich seiner Genauigkeit evaluiert, die Vorgehensweise wird diskutiert und ein Ausblick hinsichtlich der weiteren Verbesserung der Verfahren wird geboten.

Inhalt

Abkürzungsverzeichnis

TGC	Time Gain Control
AC	Alternating Current
DLR	Deutsches Zentrum für Luft- und Raumfahrt
Fraunhofer IBMT	Fraunhofer Institut für biomedizinische Forschung
KIT	Karlsruher Institut für Technologie
IPR	Institut für Prozessrechentechnik, Automation und Robotik (KIT)
MeGI	Medizingruppe am IPR
ICP	Iterative Closest Point Algorithmus
TCP	Tool-Center-Point
RAW	Rohdaten
HF	Hochfrequenz
NF	Niederfrequenz
VDI	Verein Deutscher Ingenieure
MRI	Magnetic Resonance Imaging
MRT	Magnetresonanztomographie
CT	Computertomographie
fMRT	funktionale Magnetresonanztomographie
SPECT	Single-Photon-Emission-Computed-Tomographie
PET	Positronen-Emissions-Tomographie
LWR	Lightweight Robot
LBR	Leichtbauroboter
TCP/IP	Transmission Control Protocol / Internet Protocol
PACS	Picture Archiving And Communication System
DICOM	Digital Imaging and Communications in Medicine
ISDN	Integrated Services Digital Network
CSMA/CD	Carrier Sense Multiple Access / Collision Detection
CORBA	Common Object Request Broker Architecture
API	Application Programming Interface
IGSTK	Image Guided Surgery Toolkit
VTK	Visualization Toolkit
ITK	Insight Segmentation and Registration Toolkit
CRC	Cyclic Redundancy Check
CSS	Cascaded Style Sheets
HTML	Hypertext Markup Language
GUI	Graphical User Interface
NRRD	Nearly Raw Raster Date
MRML	Multimedia Retrieval Markup Language
DiPhAS	Digital-Phased-Array-System
XML	Extensible Markup Language
FRI	Fast Research Interface
CLI	Command-Line-Interface oder Common Language Infrastructure (C++/CLI)
CLR	Common Language Runtime
CIL	Common Intermediate Language
MFC	Model-View-Controller
FPGA	Field Programmable Gate Array
CAD	Computer Aided Design

Abbildungsverzeichnis

Tabellenverzeichnis

1. Einleitung

Seit Jahrzehnten greifen Mediziner auf Ultraschallgeräte zur Schwangerschaftsverlaufskontrolle, Untersuchung von inneren Organen und allgemein zur Darstellung von Weichteilen zurück. In jedem Krankenhaus und vielen Arztpraxen stehen Ultraschallgeräte zur Verfügung. In den letzten Jahren wurde die Magnetresonanztomographie aufgrund gefallener Kosten und deutlich besserer Bildqualität immer populärer. Dennoch gilt weiterhin, dass die Anschaffung und der Betrieb von Magnetresonanztomographen wesentlich höhere Kosten verursacht, als dies bei Ultraschallgeräten der Fall ist. Gemeinsam ist beiden Modalitäten, dass sie sich vor allem zur Darstellung von Weichteilen eignen.

1.1. Motivation

Leider ist die Bildqualität eines Ultraschallbildes aufgrund von Artefakten verschiedener Ursachen, geometrischer Verzerrungen und richtungsabhängiger (relativ zur Sonde) Ortsauflösung deutlich schlechter als die der Schnittbilder, die aus MRT Geräten stammen. Zudem sind die Gesichtsfelder von Ultraschallgeräten beschränkt und dem Anwender wird sehr viel Erfahrung in der Beurteilung der gewonnenen Daten abverlangt. Deshalb wird Ultraschall bei der Beurteilung von Krankheitsbildern fast ausschließlich zur Verdachtskontrolle verwendet.

Da Ultraschallgeräte weit verbreitet und relativ kostengünstig sind, soll auch in Zukunft die Bildqualität verbessert werden, um so Ärzte und Krankenhäuser bei der medizinischen Versorgung zu unterstützen. Auch bietet Ultraschall gegenüber der Magnetresonanztomographie Vorteile bei der Handhabung der Modalitäten und der Auflösung kleinster Strukturen. Während bei der intraoperativen Verlaufskontrolle mittels MRT der Patient und der Operationstisch in das Innere des Magnetresonanztomographen bewegt werden müssen, kann ein Ultraschallgerät die anatomischen Strukturen darstellen, ohne den Patienten hierzu anders positionieren zu müssen. Hinzu kommt, dass eine Untersuchung für den Patienten mittels Ultraschall deutlich angenehmer ist, als die mittels des lauten und engen Magnetresonanztomographen.

Gerade bei der Schwangerschaftsdiagnostik bietet sich für Patientinnen die Möglichkeit, die Bewegungen des Kindes in Echtzeit zu sehen.

Um die offensichtlichen Schwächen des Ultraschalls zumindest teilweise zu kompensieren, wurden verschiedene Methoden eingeführt, die es ermöglichen die gewonnenen Daten zu verbessern, bzw. die Beurteilung der Bilder zu vereinfachen. So wurden Verfahren zur Unterdrückung des Rauschens, sowie zur Generierung von 3D-Darstellungen der untersuchten Struktur eingeführt.

Generell sehr schwierig ist die Beurteilung von Ultraschallbildern auf quantitativer Ebene. Während Strecken in den 2D-Bildern in bestimmten Bereichen sehr genau bestimmt werden können, können diese besonders im Nahbereich der Sonde deutlich von der Realität abweichen. Verfahren die 3D-Bilder erzeugen, weichen bei der Darstellung der Größe der Struktur sehr häufig von der Realität ab oder werden sehr stark durch Randbedingungen beeinflusst.

Es besteht also weiterhin Bedarf, Methoden und Technologien zu entwickeln, die die Genauigkeit, Qualität und vor allem die Benutzerfreundlichkeit der Ultraschalltechnik verbessern.

1.2. Zielsetzung

Ziel dieser Arbeit ist es, mit Hilfe vorhandener Technologien 3D-Ultraschalldatensätze von Phantomen und anatomischen Strukturen zu erstellen, die gegenüber konventionellen 3D Techniken eine verbesserte quantitative Beurteilung zulassen und eine verbesserte Qualität vorweisen. Weiterhin soll eine Entwicklungsplattform für Techniken innerhalb der noch relativ jungen Ultraschalltomographie geschaffen werden.

An festgelegten unterschiedlichen Positionen sollen 2D Ultraschallbilder erzeugt werden, denen jeweils ihre Orientierung und Position im Raum als Metadatensatz zugeordnet wird. Die so gewonnenen Daten sollen zu einem Volumendatensatz zusammengefügt werden und mittels geeigneter Methoden visualisiert werden.

Im Gegensatz zu konventionellem 3D Ultraschall sollen die Schnittbilder, aus denen der Datensatz erzeugt wird, sich ausschließlich in ihrer Translation unterscheiden.

Während es beim Freihand 3D Ultraschall unmöglich ist, keine Rotation zwischen zwei Ultraschallaufnahmen zu erzeugen, so ist dies bei Einsatz eines Roboters aufgrund der großen Genauigkeit (im Falle des Stäubli RX90 0.02mm) sehr einfach zu realisieren. Im 3D Volumensatz bleiben dadurch die 2D-Aufnahmen erhalten und können wie in CT oder MRT-Serien einzeln begutachtet werden.

Schließlich sollen verschiedene Serien mit unterschiedlichen Orientierungen erzeugt werden. Die Sonde wird hierzu um eine Achse innerhalb des Gewebes/Phatoms rotiert und weitere Datensätze werden akquiriert.

Mittels der gegeneinander registrierten Bildserien soll untersucht werden, ob und in welchem Maße Strukturen besser erkannt bzw. begutachtet werden können.

1.3. Gliederung

Zu Beginn werden in einem Grundlagenkapitel (Kapitel 2) Funktion und Aufbau medizinischer Ultraschallmodalitäten vorgestellt. Anschließend werden die allgemeinen Grundlagen zur Robotik beleuchtet.

Auf die für diese Arbeit notwendigen mathematischen Grundlagen wird ebenso eingegangen, wie auf Algorithmen und Verfahren zur 3D-Rekonstruktion und Registrierung sowie auf bildgebende Verfahren für die Medizin, die nach den Prinzipien der Tomographie arbeiten.

Kapitel 3 beschäftigt sich mit dem aktuellen Stand der Forschung. Es werden Ansätze zur 3D-Rekonstruktion mit Speckle-Dekorrelation oder externen Trackingsystemen im Arbeitsgebiet des Ultraschalls sowie Ansätze anderer Forschungsgruppen und Firmen vorgestellt, die sich ebenfalls mit Ultraschalltomographie beschäftigt haben.

Außerdem soll dieses Kapitel die noch relativ seltene Anwendung von Robotern in der Medizin beleuchten.

Zu guter Letzt stehen Projekte im Fokus, die sich mit medizinischen Echtzeitanwendungen unter Verwendung lokaler Netzwerke oder des Internets beschäftigt haben.

Kapitel 4 beschäftigt sich mit dem Aufbau und der Struktur des Ultraschalltomographen. Es wird umfassend Überblick über die verwendeten Modalitäten, sowie deren Beziehungen zu einander geboten. Verwendete Protokolle und Softwareprodukte werden beschrieben.

Um später die Ergebnisse beurteilen zu können, wird hier auch Bezug auf die Eigenschaften der Modalitäten genommen.

Kapitel 5 beschreibt detailliert die Realisierung des Ultraschalltomographen.

In Kapitel 6 werden die Arbeitsergebnisse, insbesondere Volumendatensätze erläutert und anschließend in Kapitel 7 diskutiert. An dieser Stelle wird auch die bauliche Ausführung des Tomographen diskutiert.

Kapitel 7 beschäftigt sich außerdem mit einem Ausblick auf zukünftige Möglichkeiten und Perspektiven, die sich auf Basis dieser Arbeit ergeben.

In Anhang E sind die verwendeten Soft- und Hardwareprodukte kurz beschrieben. Außerdem wird auf die Projekt- bzw. Herstellerwebsiten verwiesen.

Dieser Arbeit ist eine CD beigelegt, die Quellcode, Volumendatensätze, Online-Quellen (Stand November 2010), Ausschreibung und das vorliegende Dokument als PDF-Datei enthält. Anhang F beinhaltet das Inhaltsverzeichnis für diese CD.

2. Grundlagen

Im Folgenden werden die notwendigen physikalischen und mathematischen Grundlagen, die zur Realisierung und zum Verständnis der Ultraschalltomographie notwendig sind beschrieben. Zu Beginn wird hier näher auf Ultraschallphysik mit besonderem Augenmerk auf medizinische Bildgebung eingegangen. Danach werden wichtige Aspekte der Robotik betrachtet und die mathematischen Grundlagen zur Rekonstruktion der untersuchten Strukturen beschrieben. Den Abschluss bildet ein kurzer Abschnitt zum Thema Tomographie

2.1. Ultraschall

Ultraschall [1] wird seit vielen Jahrzehnten in den unterschiedlichsten Bereichen eingesetzt. Neben dem Einsatz in der Medizin existieren Anwendungen zur Abstandsmessung, Materialprüfung, Kommunikation, oder Materialbearbeitung. Auch Tiere wie z.B. Fledermäuse setzen Ultraschallsignale zur Kommunikation, oder Ortung ein.

Als Ultraschall werden auditiv wahrnehmbare Frequenzen oberhalb der Hörschwelle des Menschen bezeichnet. In der Regel werden Frequenzen zwischen 20 kHz und 1.6 Ghz zum Ultraschall gezählt. Noch höhere Frequenzen werden als Hyperschall bezeichnet. Für die medizinische Bildgebung werden vor allem Frequenzen zwischen 1 und 40Mhz verwendet.

Für Ultraschall gelten wie auch bei hörbarem Schall Reflexions-, Transmission-, Streuungs-, und Brechungsbedingungen in verschiedenen Materialen und an verschiedenen Materialkanten.

Die beschriebenen Effekte sind in Abbildung 1 illustriert.

Diese unterschiedlichen Materialeigenschaften werden bei der Sonographie zur Darstellung von Gewebegrenzen oder Strukturen innerhalb des Gewebes verwendet.

Unterschiede ergeben sich aufgrund unterschiedlicher Dichte und unterschiedlicher Schallschnelle (oder Fortleitungsgeschwindigkeit).

[1] Bildverarbeitung 2 Skript 2010 R.Bendl Medizinische Informatik Heilbronn/Heidelberg S. 272-320

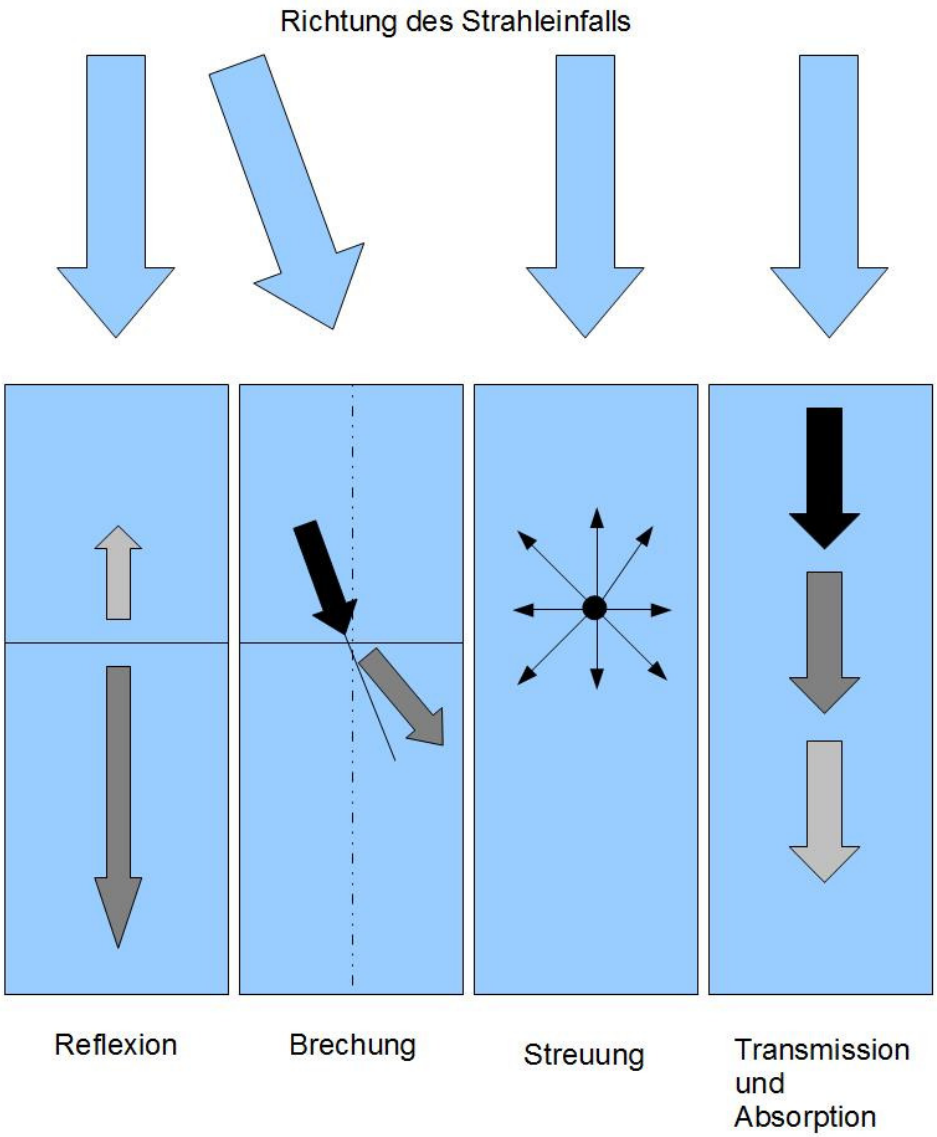

Abbildung 1: Ultraschalleffekte im Gewebe

Die Impedanz Z ist das Produkt aus Schalldruck ρ und Schallschnelle v. Die Schallschnelle ergibt sich hierbei wie bereits erwähnt durch die Gewebeeigenschaften. Der Schalldruck hingegen ist abhängig von der Intensität des eingestrahlten Schalles. Es ergibt sich

$$Z = \frac{\rho}{v}$$

Formel 1: Abhängighkeit von Impedanz, Schalldruck und Schallschnelle

Tabelle 1 stellt Schallschnellen, Impedanzen und Dichten verschiedener Materialien gegenüber.

	$C[\frac{m}{s}]$	$Z[g * cm^2 * c]$	$\rho[\frac{g}{cm^3}]$
Luft	331	43	0.013
Wasser	1492	$1.48 * 10^5$	0.9982
Fett	1470	$1.42 * 10^5$	0.97
Hirn	1530	$1.56 * 10^5$	1.02
Muskel	1568	$1.63 * 10^5$	1.04
Knochen	3600	$6.12 * 10^5$	1.7

Tabelle 1: Schalleigenschaften verschiedener Materialien[2]

Aufgrund der Impedanzunterschiede ergeben sich Reflexionen die von der Ultraschallsonde gemessen werden können. An Grenzschichten zwischen Geweben kommt es zu Impedanzsprüngen und somit zu einer Reflexion eines Anteiles der Ultraschallwellen.

Um den Anteil des reflektierten Schalls zu bestimmen wird der berechnet Reflexionskoeffizienten R berechnet. R ist das Verhältnis zwischen reflektierter und einfallender Schallintensität.

R wird wie folgt berechnet:

$$R = \frac{Z_2 - Z_1}{Z_2 + Z_1}$$

Formel 2: Berechnung des Reflexionskoeffizienten anhand verschiedener Impedanzen in Gewebetypen

Nun wird auch deutlich, weshalb Knochen und Luft die Schallfortleitung negativ beeinflussen. Die Impedanzen von Knochen und Luft unterscheiden sich sehr stark von den Impedanzen im Gewebe. Dadurch ergeben sich extrem große Reflexionskoeffizienten. Im Falle von Luft beträgt der Reflexionskoeffizient circa 99% bei Ankopplung an Weichgewebe. Dies bedeutet dass 99% des einfallenden Schalles als Reflexion zum Schallkopf reflektiert werden. Um den Schall zwischen Sonde und Gewebe einzukoppeln wird deshalb ein Gel auf Wasserbasis verwendet, welches im Vergleich zum Gewebe geringe Impedanzunterschiede aufweist. Treffen die Ultraschallwellen im Gewebe auf Knochen, so wird der Reflexionskoeffizient ebenfalls sehr groß (ca. 58%) Deshalb ist die Schallintensität im und hinter dem Knochen stark reduziert und die Darstellungsqualität leidet.

Zur Erzeugung der Ultraschallwellen werden in Ultraschallsonden Piezokristalle verwendet, die durch den inversen piezoelektrischen Effekt in Schwingung versetzt werden. Der piezoelektrische Effekt beschreibt die Ladungsverschiebung innerhalb von Festkörpern, die bei Einwirkung äußerer Kräfte und damit elastischer Verformung des Festkörpers einhergeht. Als direkter Piezoeffekt wird die Einwirkung mechanischer Kräfte bezeichnet, die einen Festkörper elastisch verformen und so einen Dipol erzeugen. Hierbei entstehen messbare Spannungen am Piezokristall. Der umgekehrte Effekt wird bei der Erzeugung des Ultraschalls verwendet. Es handelt sich hierbei um den inversen piezoelektrischen Effekt. Hierbei wird an einen Piezokristall eine Spannung angelegt und somit eine elastische Verformung ausgelöst.

[2] Medizinphysik Skript – Ultraschall, Dieter Suter 2007, Universität Dortmund

Bei der Sonografie wird zur Spannungserzeugung ein Hochfrequenzgenerator verwendet. Das am Kristall anliegende Wechselfeld bringt den Kristall zum oszillieren. Diese Oszillation wird als Ultraschall auf das Medium vor der Sonde übertragen.

Zur Messung des Ultraschalls wird der direkte piezoelektrische Effekt verwendet. Ultraschallwellen bewirken eine elastische Verformung am Kristall, welche eine Spannungsänderung zur Folge hat, die vom Ultraschallgerät gemessen werden kann.

Eine Sonde kann also sowohl als Empfänger, als auch als Sender dienen.

Bei den meisten Ultraschallgeräten wird wie in Abbildung 2 dargestellt das so genannte Echo-Impuls Verfahren verwendet. Hierbei dient ein und dieselbe Sonde sowohl als Sender, als auch als Empfänger. Ein Taktgeber wird verwendet, um das vom Hochfrequenzgenerator erzeugte Signal immer nur zu diskreten Zeitpunkten in die Kristalle einzuspeisen. Hierbei entsteht durch einen Impuls Ultraschall, der im Gewebe reflektiert, absorbiert und transmittiert wird. Zum Kristall zurückreflektierte Schallwellen bewirken den inversen piezoelektrischen Effekt, werden über eine Verarbeitungselektronik gemessen und in geeigneter Signalform dargestellt. Die Amplitude der Spannung ist abhängig von der Intensität der reflektierten Komponente des Ultraschalls und so direkt abhängig von den Impedanzunterschieden an Gewebegrenzflächen oder im Gewebe.

Die Zeitdauer zwischen gesendetem Impuls und empfangenem Echo beschreibt die Entfernung der reflektierenden Schicht. Das Ultraschallgerät nimmt hierzu eine Wellenausbreitungsgeschwindigkeit von 1540m/s im Gewebe an. Aufgrund der unterschiedlichen Fortleitungsgeschwindigkeiten im Gewebe entstehen hierbei Verzerrungen, die Ursachen von geometrischen Verzerrungen in Ultraschallbilder sind.

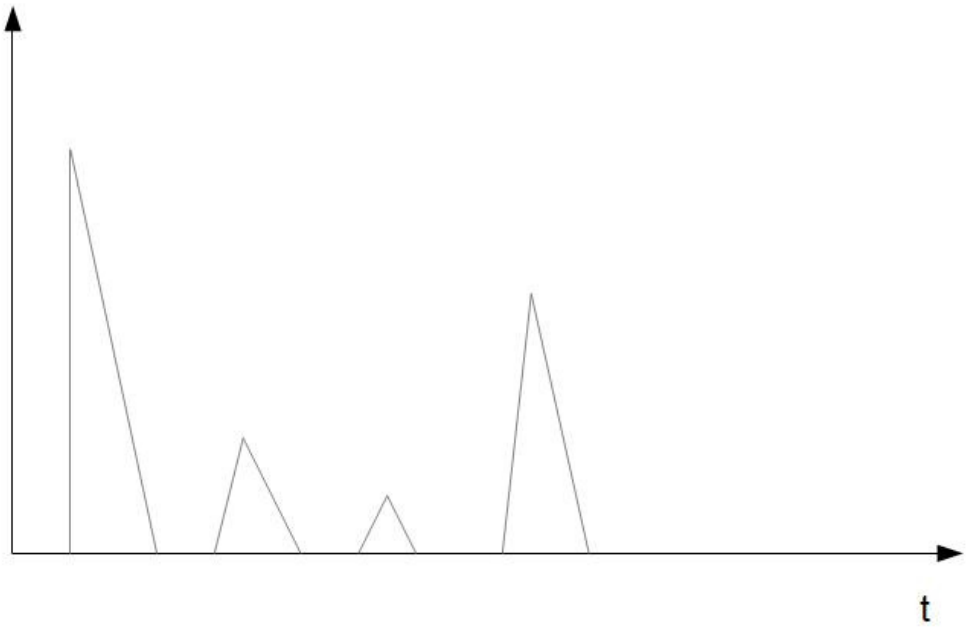

Abbildung 2: Echo-Impulsverfahren. Der erste Impuls mit senkrechter aufsteigender Flanke ist Sendeimpuls. Alle weiteren Impulse sind an Grenzflächen reflektierte Schallanteile

Andererseits können aber auch getrennte Sonden verwendet werden, die sich auf den gegenüberliegenden Seiten des zu schallenden Objektes befinden. Sonde 1 wird hierbei zur Signalerzeugung verwendet. Sonde 2 misst im Gegensatz zum Echo-Impuls Verfahren transmittierten Ultraschall. Dieses Verfahren wird in ähnlicher Form auch bei der Röntgencomputertomografie verwendet, bei der sich das zu untersuchende Objekt zwischen Röntgenröhre (Sonde 1) und Röntgendetektor (Sonde 2) befindet.

Das während dieser Diplomarbeit eingesetzte Ultraschallgerät arbeitet nach dem Echo-Impuls Verfahren und benötigt deshalb nur eine Ultraschallsonde.

Moderne Ultraschallwandler (Arraysonden) bestehen aus mehreren nebeneinanderliegenden Piezokristallen. Abbildung 3 zeigt die unterschiedlichen Funktionsweisen von Arrayschallsonde und Wobbler. Der Wobbler verwendet im Gegensatz zur Arrayschallsonde einen einzelnen Piezokristall der mechanisch in der Bildebene bewegt wird. Arraysonden werden deutlich häufiger als Wobbler verwendet.

Durch den Einsatz mehrerer Kristalle und durch Verwendung einer zeitversetzung Ansteuerung ergibt sich die Möglichkeit Ultraschallwellen zu fokussieren und damit die Auflösung lokal zu erhöhen. Außerdem bieten sich Vorteile bei der Geschwindigkeit der Bildgebung. Mehrere Strahlen werden gleichzeitig ausgesendet und empfangen. Durch die nahezu parallelen Strahlen kann dabei mittels eines einzigen Scans ein 2D-Schnittbild erzeugt werden. Berücksichtigt werden müssen Wellen, die nicht an senkrechten sondern an schrägen (weder horizontal, noch vertikal relativ zur Sonde) Kanten

im Gewebe reflektiert wurden und somit nicht denselben Kristall treffen, von dem sie ausgesendet wurden, sondern einen der anderen Kristalle in der Sonde.

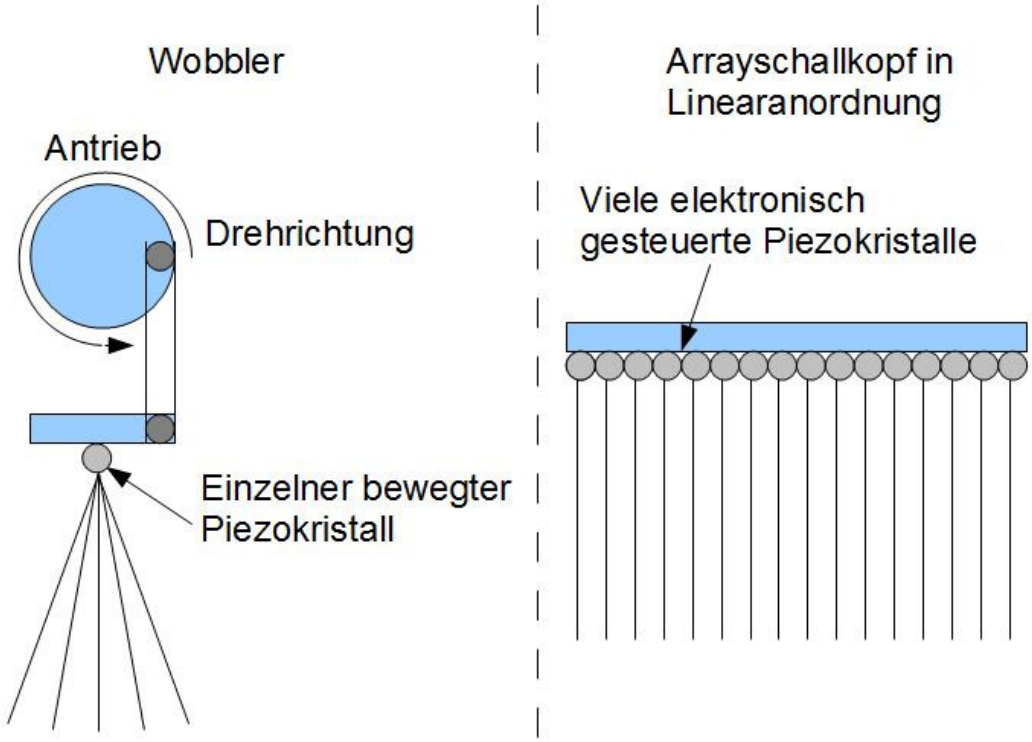

Abbildung 3: Funktionsweise von Wobbler und Arrayschallkopf

Analog zu Punktstrahlern, die hörbaren Schall aussenden, gibt es auch bei Ultraschallwandlern Nah- und Fernfeld. Im Nahfeld einer Schallquelle herrschen starke Interferenzen und inhomogene Intensitätsverteilung. Je weiter ein Objekt sich von der Sonde entfernt und somit ins Fernfeld gerät, desto homogener wird die Intensitätsverteilung, an seiner Position. Die Interferenzen nehmen ab.

Deshalb und aufgrund der Eindringtiefe müssen für unterschiedliche zu untersuchende Körperteile verschiedene Frequenzen und somit Sonden verwendet werden. Wegen der Resonanzfrequenz der Piezokristalle können diese nur in einem eng begrenzten Frequenzraum schwingen. Daraus folgt, dass nicht alle Sonden für jedes Körperteil und jede Frequenz einsetzbar sind.

Je geringer die Frequenz, desto größer ist das Nahfeld eines Ultraschallwandlers und desto weniger Strahlung wird im Gewebe absorbiert. Die Eindringtiefe eines Wandlers sinkt also mit steigender Frequenz der Ultraschallwellen. Je größer die Frequenz der Schallwellen, desto geringer ist die Wellenlänge. Die Lokalisation einer Brechkante im Gewebe kann also mit sinkender Wellenlänge genauer festgestellt werden. Je geringer die Frequenz, desto größer ist die Eindringtiefe einer Ultraschallsonde und desto geringer die Ortsauflösung.

Für Untersuchungen im Halsbereich, wo nur eine geringe Eindringtiefe, aber hohe Auflösung benötigt werden, eignen sich 7.5 Mhz Sonden. Für Untersuchungen im Bauchraum, wo große Eindringtiefen benötigt werden, kommen vor allem Sonden im Bereich von 3.5Mhz zum Einsatz.

Die im Impuls-Echo empfangenen Signale bilden allerdings noch keine Bilder, sondern sind lediglich Rohdaten (RAW) Daten, die in mehreren Schritten zuerst in Hochfrequenzdaten pro Ultraschallarray und anschließend nach Logarithmierung in Grauwerte übersetzt werden, die die Intensitäten des reflektierten Schalles repräsentieren.

Die empfangenen und eventuell nachverarbeiteten Rohdaten der Piezokristalle werden für den A-Mode des Ultraschallgerätes verwendet. Auf der X-Achse eines A-Mode Scans ist die Zeit bis zur Rückkehr des Schalles aufgetragen. Auf der Y-Achse ist die Intensität des reflektierten Schalles aufgetragen. Ein A-Mode Diagramm stellt die Druckverhältnisse an einem einzelnen Piezokristall in Abhängigkeit von der Zeit dar. Die in Grauwerte umgesetzten HF Daten hingegen werden für die B-Mode Bildgebung verwendet, dessen Bilder für Menschen sehr viel einfacher zu interpretieren sind.

Abbildung 4 zeigt ein B-Mode Bild der Schilddrüse.

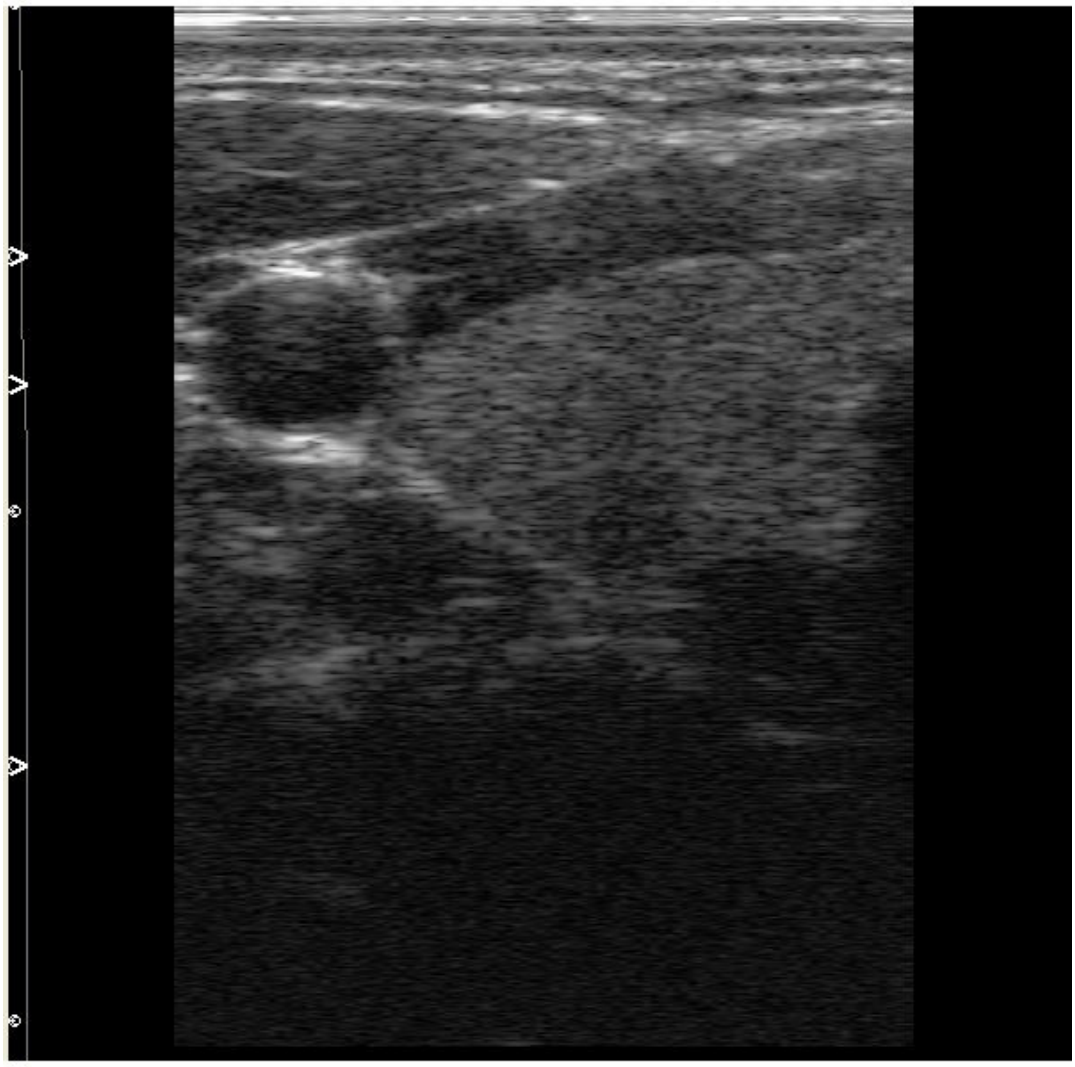

Abbildung 4: B-Mode Bild des linken Schilddrüsenlappens, des Isthmus und der linken Arteria Carotis des Autors

Moderne Ultraschallgeräte bieten außerdem noch weitere Modi wie M-Mode oder Farbdoppler, auf die hier nicht näher eingegangen wird.

Wie bereits erwähnt kommt es durch unterschiedliche Schallgeschwindigkeiten in unterschiedlichen Medien zu Verzerrungen in den Ultraschalldaten, die im ungetrackten 2D Ultraschall mit konventionellen Methoden kaum berücksichtigt werden können.

Die Darstellungsqualität in Ultraschallbildern wird allerdings noch durch weitere physikalische Gegebenheiten beeinträchtigt. So verursacht die Reflexion der Wellen an rauen Kanten gewebespezifische Muster, die als für Ultraschallbilder charakteristische Specklemuster wahrgenommen werden. Diese für die Wahrnehmung negativ zu bewertenden Speckles können im Freihand 3D-Ultraschall verfolgt und somit zur Erstellung von 3D Darstellungen verwendet werden, ohne dass externe Trackingsysteme benötigt werden.

Hinter stark reflektierenden Objekten kann es wie in Abbildung 5 zu sehen zu einer distalen Schallauslöschung kommen. Die tieferliegenden Strukturen werden schlechter dargestellt. Diesem Phänomen kann begrenzt mittels eines von der Eindringtiefe abhängigen Verstärkungsfaktors (Time Gain Correction) begegnet werden, der aber vor allem zur Kompensation der in der Tiefe schwächer werdenden Wellen genutzt werden kann. Der gegenteilige Effekt, der hinter schwach dämpfenden Medien auftritt, wird distale Schallverstärkung genannt. Als laterale Auslöschung werden Artefakte bezeichnet, die durch Reflektion des Schalles an gekrümmten Objekten entstehen. Hierbei werden die Anteile der dort auftreffenden Strahlung abgelenkt und tragen so nicht mehr zur Bildgebung hinter dem Objekt bei.

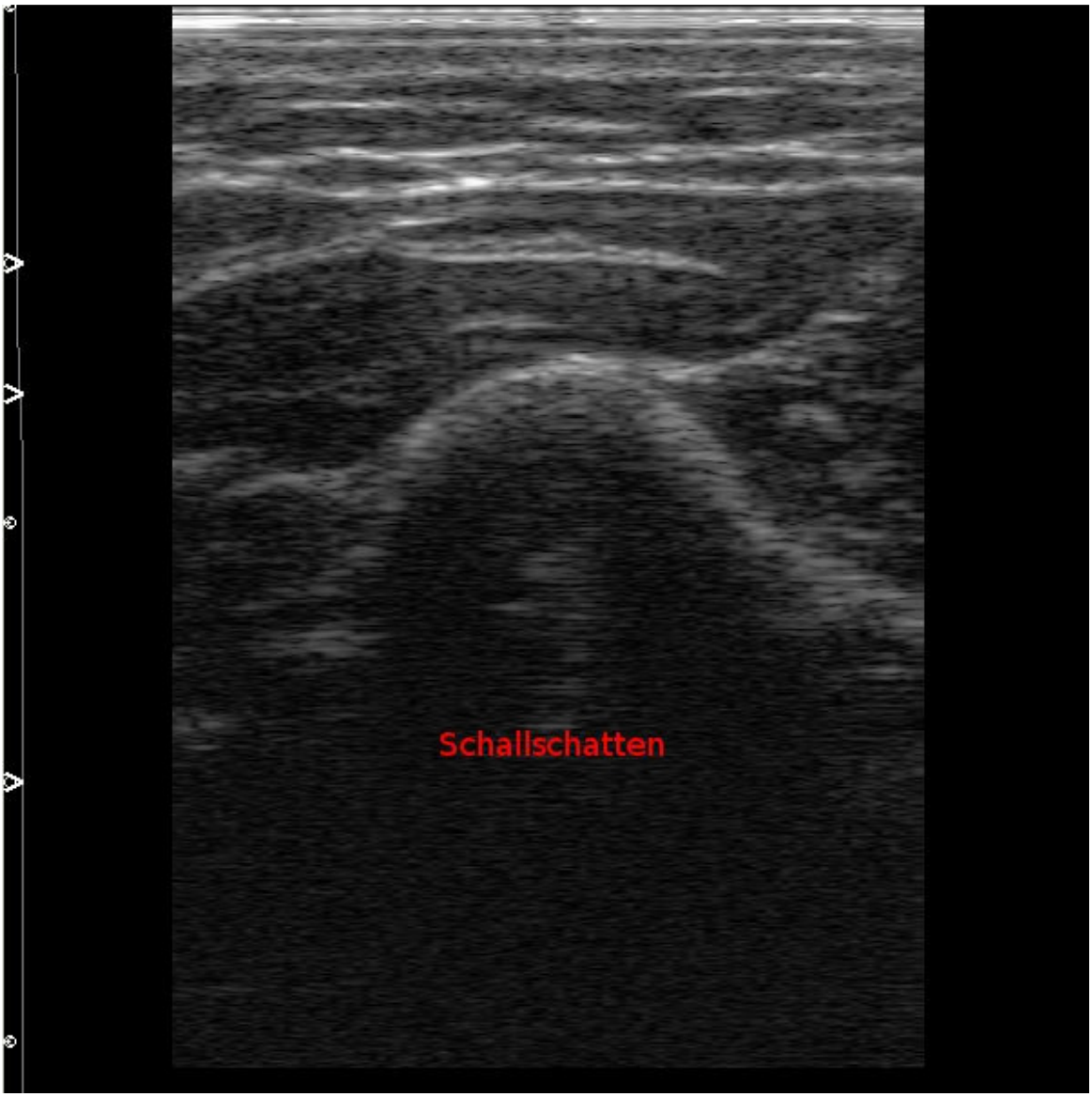

Abbildung 5: distale Schallauslöschung hinter der Grenzfläche zwischen Weichgewebe und Radius (Speiche)

Schließlich können bei der Reflexion des Schalles an stark reflektierenden Oberflächen (z.B. Metalle) Kometenschweifartefakte auftreten. Diese Artefakte sind Mehrfachreflexionen, die zu Scheinreflexionen im Ultraschallbild führen.

Starke Reflexionen am Boden eines mit Wasser gefüllten Glasbeckens sind in Abbildung 6 zu sehen.

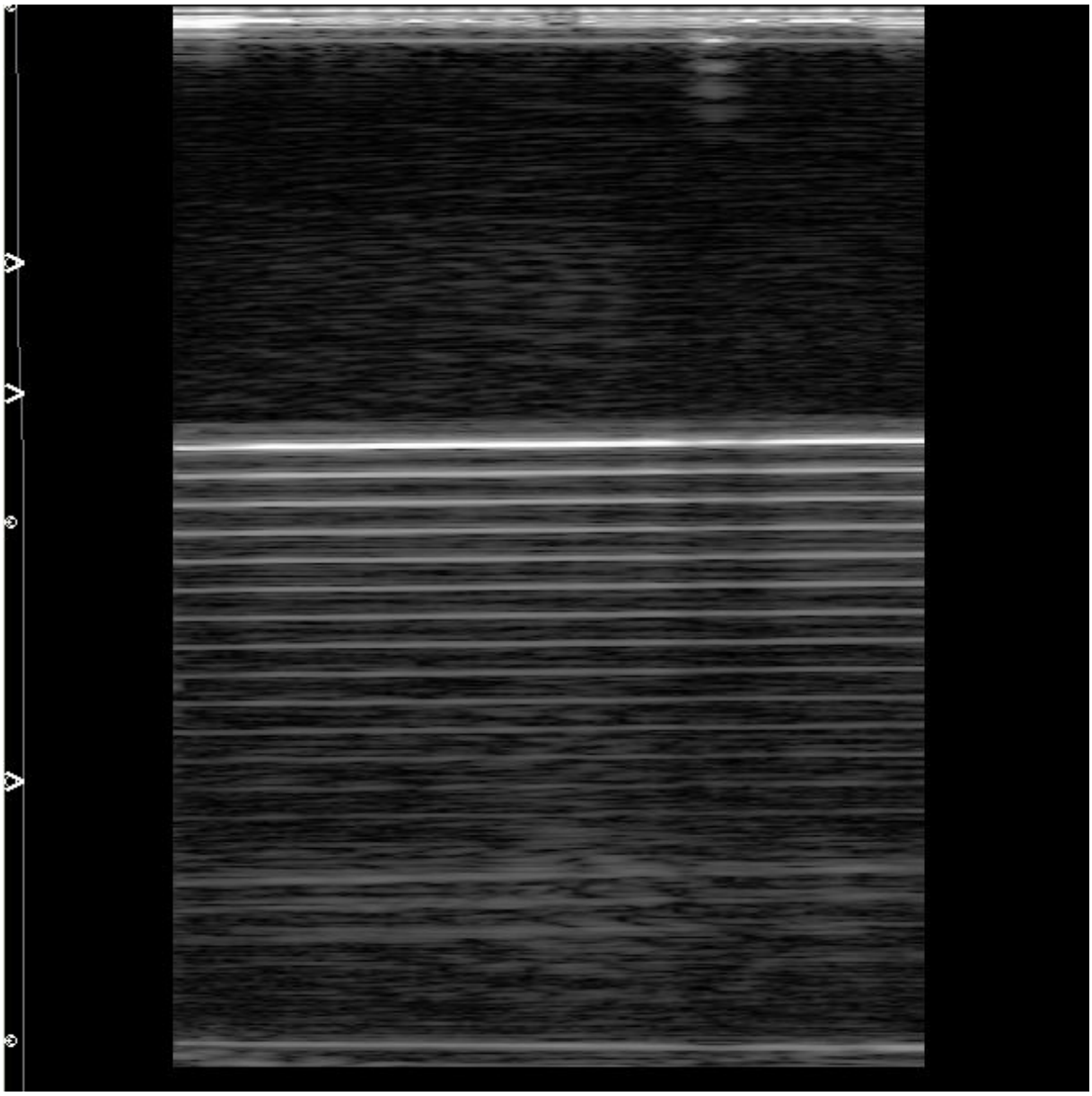

Abbildung 6: Starke Reflexionen mit Geisterbildern am Boden eines mit Wasser gefüllten Glasbeckens

Ultraschallbilder weisen also sowohl geometrische Verzerrungen durch Schallgeschwindigkeitsunterschiede, Rauschen durch Speckles, als auch Artefakte durch Schallauslöschungen oder Verstärkungen auf. Damit nicht genug weist die Schallkeule eine von der Eindringtiefe abhängige Breite auf, die ebenfalls zu geometrischen Verzerrungen führt, die im Bereich des Fokuspunktes am geringsten sind.

Das Auflösungsverhalten einer Ultraschallsonde ist wie bereits erwähnt abhängig von der Frequenz des Ultraschalls. Außerdem unterscheidet sich die axiale Auflösung von der lateralen Auflösung. Diese ist aufgrund der im letzten Absatz erwähnten Strahlcharakteristik im Gegensatz zur axialen Auflösung nicht ausschließlich von der Wellenlänge abhängig und nimmt mit der Eindringtiefe ab.

Typische Werte für die axiale Auflösung liegen im Bereich von $2*\lambda$ und zwischen $4\text{-}5*\lambda$ für die laterale Auflösung.

2.2. Robotik

Roboter sind in vielen Bereichen sinnvoll einsetzbar. So sind sie in der Industrie und insbesondere im Kraftfahrzeugbau ein unverzichtbares Mittel, um mit hoher Effizienz, Geschwindigkeit und Genauigkeit arbeiten zu können.

Nach VDI-Richtlinie 2860[3] sind Industrieroboter wie folgt definiert:

„Industrieroboter sind universell einsetzbare Bewegungsautomaten mit mehreren Achsen, deren Bewegungen hinsichtlich Bewegungsfolge und Wegen bzw. Winkeln frei (d.h. ohne mechanischen Eingriff) programmierbar und ggf. sensorgeführt sind. Sie sind mit Greifern, Werkzeugen oder anderen Fertigungsmitteln ausrüstbar und können Handhabungs- und/oder Fertigungsaufgaben ausführen“
(VDI 2860)

Roboter existieren in den unterschiedlichsten Bauformen. Hier wurden ausschließlich Knickarmroboter mit 6 bzw. 7 Achsen verwendet, die sich durch große Arbeitsräume und gute Erreichbarkeit der Punkte im Raum auszeichnen.

Häufig verwendet werden auch Portalroboter, SCARA Roboter und Roboter mit parallelen Kinematiken wie Hexapoden oder Delta Roboter[4].

Industrieroboter wie der verwendete Stäubli RX90 benötigen mindestens 6 Freiheitsgrade um theoretisch jeden Punkt in ihrem Arbeitsraum in jeder möglichen Orientierung anzufahren.

Es werden hierbei drei Translationen in X, Y und Z Richtung, sowie drei Rotationen um X, Y und Z-Achse unterschieden.

Mindestens benötigt zur technischen Realisierung eines Industrieroboters werden:

- Steuerung
- Antriebe
- Sensorik
- Kinematik

Die Steuerung überwacht hierbei alle Bewegungen des Roboters, misst aktuelle Gelenkpositionen und bewegt über die Antriebe die Gelenke der Roboter. In der Regel überwacht die Robotersteuerung auch technische Sicherheitseinrichtigungen, wie geschlossene Türen eines Sicherheitskäfigs oder den Druck des Totmannschalters durch den Anwender. Außerdem wird über die Steuerung die Anbindung zur Außenwelt hergestellt. Dies wird häufig über die serielle RS232 Schnittstelle oder verschiedene Netzwerkprotokolle realisiert.

Als Antriebe werden die Einheiten bezeichnet, die die einzelnen Teile der Kinematik bewegen. Hierzu gehören Motoren, Getriebe und Regelung der Motoren.

Die interne Sensorik des Roboters überwacht in den Gelenken deren Positionen. Die Sensorik wird benötigt um festzustellen, in welcher Position sich die einzelnen Robotergelenke aktuell befinden.

[3] Montage- und Handhabungstechnik; Handhabungsfunktionen, Handhabungseinrichtungen; Begriffe, Definitionen, Symbole; VDI 2860
[4] Medizinische Robotik Skript 2009 J.Raczkowsky Institut für Prozessrechentechnik Automation und Robotik

Die Steuerung verwendet die aktuellen Positionen (Ist-Position), um die Antriebe so zu regeln, dass die gewünschte Position (Soll-Position) erreicht wird.

Bei der Kinematik handelt es sich um die Teile der Mechanik, die alle Kräfte und Lasten tragen, die auf den Roboter wirken (inklusive des Eigengewichts). Die Kinematik realisiert die eingangs erwähnten Bewegungsachsen und ist über ihre Geometrie eindeutig beschrieben. Abbildung 7 zeigt einen typischen industriell verwendeten Roboter der Firma Kuka.

Abbildung 7: Industrieroboter der Firma Kuka in der Roboterhalle des IPR

In dieser Arbeit wurden Roboter vor allem als kartesische Positionierungsgeräte verwendet. Dies bedeutet, dass der Roboter mit dem montierten Werkzeug einen vorgegebenen Punkt im kartesischen Raum anfährt und das Werkzeug mit größtmöglicher Genauigkeit dort positioniert.

Hierbei ergibt sich die Problematik, dass der Roboter selbst ausschließlich über Drehgelenke definiert ist und die Position im kartesischen Raum nur über die Kinematik bestimmt wird. Außerdem unterscheiden sich die Koordinatensysteme an der Basis des Roboters und dem Endeffektor des Roboters, der bei nicht montiertem Werkzeug häufig am Flansch des Roboters angenommen wird und bei montiertem Werkzeug an dessen Spitze. Um eine Zuordnung der Gelenkpositionen, die in Winkelgraden angegeben oder gelesen werden können, zu einem kartesischen Koordinatensystem zu ermöglichen, wird eine für jede Robotergeometrie individuelle Vorwärtskinematik, bzw. inversen Kinematik verwendet.

Die Vorwärtskinematik realisiert hierbei über die geometrischen Gegebenheiten die Abbildung der Gelenkwinkel auf eine kartesische Position im Raum. Hierzu wird in den meisten Fällen die Denavit-Hartenberg-Transformation verwendet.

Die inverse Kinematik beschreitet den umgekehrten Weg und berechnet die notwendigen Gelenkwinkel, um eine bestimmte Position im kartesischen Raum zu erreichen. Wegen Uneindeutigkeiten an bestimmten Positionen im Raum (je nach Geometrie des Roboters) ist die Berechnung oft nicht einfach zu realisieren.

Abbildung 8 zeigt schematisch die Kinematik eines Roboters und die Koordinatensysteme einzelner Elemente des Roboters.

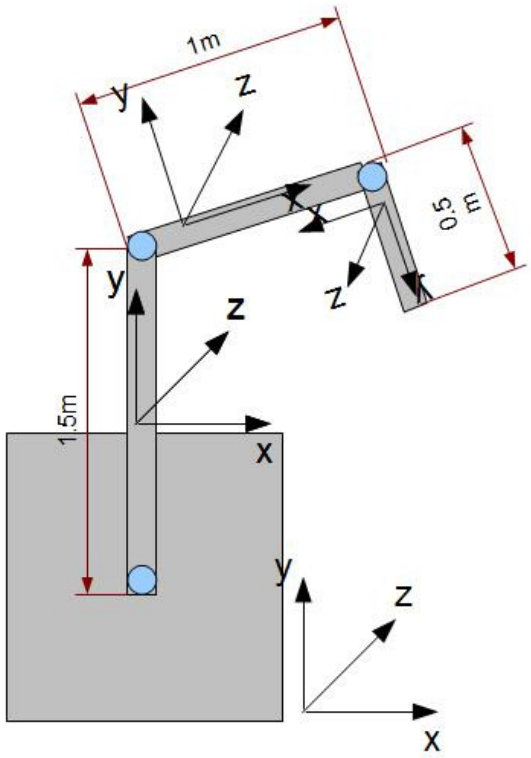

Abbildung 8 Beispiel einer Roboterkinematik mit lokalen Koordinatensystemen

Um die Position eines Werkzeuges, Werkstückes oder des Roboters relativ zum Raum auszudrücken, reicht es jedoch nicht aus ausschließlich in einem einzigen Koordinatensystem zu arbeiten.

Es werden in der Regel

Weltkoordinatensystem,
Basiskoordinatensystem
und Werkzeugkoordinatensystem

sowie weitere für ein individuelles Problem benötigte Koordinatensysteme unterschieden.

Das Weltkoordinatensystem dient als Referenz für alle anderen Koordinatensysteme. Sein Ursprung liegt meist in der Basis des Roboters, also im Zentrum der ersten Rotationsachse, kann sich aber je nach Anwendung an einem beliebigen frei definierbaren Punkt im Raum befinden. Sinnvoll ist das Positionieren des Ursprungs außerhalb der Roboterbasis zum Beispiel bei Verwendung mehrerer Roboter, die sich abhängig voneinander bewegen.

Der Ursprung des Basiskoordinatensystems eines Roboters liegt in der Basis, also dem ortsfesten Punkt eines Roboters. Es ist sinnvoll dieses als Referenz für Inverse- und Vorwärtskinematik zu wählen, da die geometrischen Verhältnisse zwischen Roboterflansch und Basis über die Kinematik eindeutig bestimmt sind.

Der Ursprung des Werkzeugkoordinatensystems befindet sich in der Werkzeugspitze. Das Werkzeugkoordinatensystem bezieht sich auf das Basiskoordinatensystem eines Roboters und gibt relativ zu diesem die Position und die Orientierung des Werkzeuges an.

Um den Roboter an eine gewünschte Position zu fahren, wird diese in Werkzeugkoordinaten relativ zum Basis- oder Weltkoordinatensystem angegeben. Anschließend wird über Methoden der linearen Algebra (siehe Mathematische Grundlagen) die zugehörige Position und Orientierung des Roboterflansches in Basiskoordinaten berechnet. Die inverse Kinematik berechnet dann zu Position und Orientierung die benötigten Gelenkstellungen, die schließlich an die Steuerung des Roboters gesendet werden.

Abbildung 9 illustriert die verschiedenen Koordinatensysteme, bei Verwendung eines Roboters.

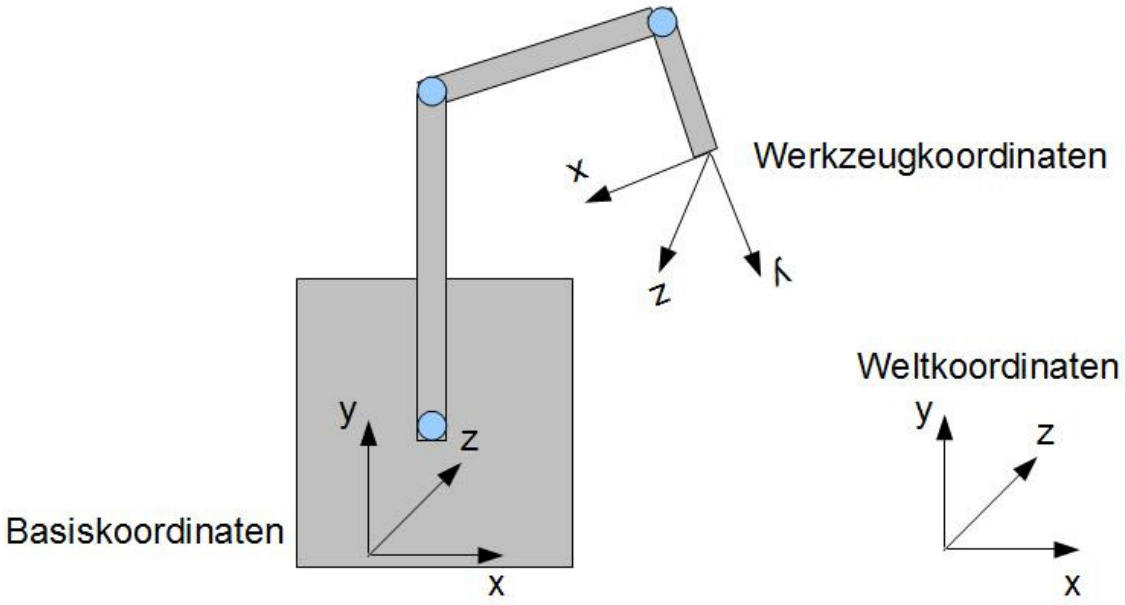

Abbildung 9: Koordinatensysteme bei Verwendung eines Roboters

2.3. Mathematische Grundlagen

Während dieser Arbeit wurden Orientierungen und Positionen von Objekten im Raum ausschließlich in homogenen Koordinaten dargestellt[5]. Des Weiteren wurde mehrfach von einem Koordinatensystem in ein Anderes gewechselt, um die Koordinaten sowohl relativ zur Sonde, als auch zum Roboter oder zur Welt darstellen zu können. In Kapitel 2.3 wird deshalb auf homogene Koordinaten und deren Anwendung sowie dem Wechseln der Basis zur Transformation in ein anderes Koordinatensystem eingegangen.

Eine Rotationsmatrix im dreidimensionalen Raum ist über 3 Spalten und 3 Zeilen definiert, die die Rotationen um X, Y und Z Achse enthalten.

Wir legen folgendes lineares Koordinatensystem zugrunde

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} * \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

Formel 3: Gleichungssystem zum rotieren eines Vektors

Der Vektor $\begin{pmatrix} x \\ y \\ z \end{pmatrix}$ wird mit einer Rotationsmatrix multipliziert, die die Koeffizienten zur Rotation um den Urpsrung enthält. Hieraus ergibt sich der neue Vektor $\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix}$ der den Ausgangspunkt im Raum im um die Matrix $\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix}$ rotierten Koordinatensystem darstellt[6].

Mittels dieser Transformation kann ein Basiswechsel durchgeführt werden. Der umgekehrte Weg lässt sich ausgehend von

$$A = B * D * B^{-1}$$

Formel 4: Grundlage zum umkehren einer Transformation

über die Multiplikation

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix}^{-1} * \begin{pmatrix} x' \\ y' \\ z' \end{pmatrix}$$

Formel 5: Umkehr einer Transformation

berechnen.

Notwendige Eigenschaften von Rotationsmatrizen sind weiterhin die Orthogonalität und die gemeinsame Determinante +1 aller Rotationsmatrizen.

[5] Bildverarbeitung 2 Skript 2010 Maier-Hein Medizinische Informatik Heilbronn/Heidelberg S.165-S.170
[6] Lineare Algebra Skript 2007 Laun Medizinische Informatik Heilbronn/Heidelberg

Aufgrund der Orthogonaliät gilt folgende Bedingung:

$$B^{-1} = B^T$$

Formel 6: Gleichheit von Transponierter- und Invertierter Matrix bei orthogonalen Matrizen

und damit:

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix}^T * \begin{pmatrix} x' \\ y' \\ z' \end{pmatrix}$$

Formel 7: Ausnutzung von Formel 8 zur Umkehr einer Transformation

Die Transponierte ist sowohl einfacher zu berechnen als die Inverse einer Matrix (geringerer Rechenaufwand), als auch numerisch stabiler.

Die Transponierte selbst ist definiert als:

$$B * B^T = B^T * B = I$$

Formel 8: Definition der transponierten Matrix

Wobei **I** die Einheitsmatrix ist.

Wir definieren nun die Rotationen um die einzelnen Achsen, um die Koeffzienten $a_{i,j}$ zu bestimmen.

Bei jeder Rotation um eine Achse handelt es sich um eine orthogonale Transformation. Die Rotationen sind wie folgt definiert:

Rotation um X-Achse:

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\varphi & -\sin\varphi \\ 0 & \sin\varphi & \cos\varphi \end{pmatrix} * \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

Formel 9: Rotation eines Vektors um X-Achse des Koordinatensystems

Rotation um Y-Achse:

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} \cos\varphi & 0 & \sin\varphi \\ 0 & 1 & 0 \\ -\sin\varphi & 0 & \cos\varphi \end{pmatrix} * \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

Formel 10: Rotation eines Vektors um Y-Achse des Koordinatensystems

Rotation um Z-Achse:

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} \cos\varphi & \sin\varphi & 0 \\ -\sin\varphi & \cos\varphi & 0 \\ 0 & 0 & 1 \end{pmatrix} * \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

Formel 11: Rotation eines Vektors um Z-Achse des Koordinatensystems

Wobei φ der Eulersche Winkel ist, um den um die betreffende Achse gedreht wird. Rotationen lassen sich beliebig über Matrixmultiplikationen verknüpfen. Dementsprechend ergibt sich als Matrix mit allen $\varphi = 0$ die Einheitsmatrix.

Mittels Rotationsmatrizen können Orientierungen von Koordinatensystemen und Objekten im Raum angegeben werden. Sie bilden auch die Basis für die Steuerung eines Industrieroboters.

Es können Koordinaten von Objekten im Raum in beliebige Koordinatensysteme transformiert und dort Translationen vorgenommen werden. So können Roboter ohne viel Aufwand und unabhängig von der Orientierung des Werkzeugs auf der X, Y, oder Z-Achse des Werkzeuges verfahren oder um einen fiktiven Punkt im Raum gedreht werden.

Bei Verwendung dreidimensionaler Rotationsmatrizen und der gewünschten Translation müssen diese immer getrennt voneinander betrachtet werden. Um dieses Manko zu beheben ist es möglich Rotation und Translation zu einer gemeinsamen Matrix zu kombinieren. Dies wird „Übergang zu homogenen Koordinaten“ genannt.

Beim Übergang zu homogenen Koordinaten wird der Translationsvektor um das Element **w** erweitert.

Üblicherweise wird zur Homogenisierung w=1 verwendet. Es ergibt sich also

$$\begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

Damit muss die Rotationsmatrix ebenfalls um eine zusätzliche Zeile erweitert werden. Zusammen mit der Translation ergibt sich

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & x \\ a_{21} & a_{22} & a_{23} & y \\ a_{31} & a_{32} & a_{33} & z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Formel 12: Repräsentation von Rotation und Translation in homogenen Koordinaten

als homogene Repräsentation für Rotation und Translation.

Homogene Koordinaten sind in der Robotik der Standard um Werkzeugkoordinaten im Verhältnis zu Roboterbasiskoordinaten (oder anderen Koordinatensystemen) anzugeben. Beide verwendeten Roboter können homogene Koordinaten sowohl entgegennehmen und anfahren, als auch die Position des Flansches in homogenen Koordinaten liefern.

2.4. Registrierung und 3D Rekonstruktion

Registrierung und dreidimensionale Rekonstruktionen sind für die medizinische Bildgebung von enormer Wichtigkeit.

Methoden der Registrierung beschäftigen sich damit zwei oder mehrere Bildserien im Raum oder im Verhältnis zueinander richtig zu positionieren und zur orientieren.

3D Rekonstruktionen ermöglichen es, aus einer Bildserie zweidimensionaler Bilddaten eine dreidimensionale Rekonstruktion der Objekte zu generieren, die die Bildserie repräsentieren.

Der Vollständigkeit halber muss an dieser Stelle auch die Segmentierung genannt werden, die sich mit der Erkennung und Abgrenzung von Objekten in 2D- und 3D-Datensätzen oder auch Videos beschäftigt. Methoden der Segmentierung kamen während dieser Arbeit nicht zum Einsatz, sind aber für eine mögliche Weiterverarbeitung der gewonnen Datensätze von Bedeutung.

2.4.1. Registrierung

Da die Registrierung aufgrund der jederzeit bekannten Sondenposition in dieser Arbeit eine eher untergeordnete Rolle spielt, soll sie nur kurz beleuchtet werden.

Wie bereits erwähnt spielt die Registrierung eine Rolle, sobald mehrere Bildserien gegeneinander orientiert werden sollen. Wichtig ist dies bei der Verlaufskontrolle mittels CT und MRT, bei der zwei Bildserien gegeneinander registriert werden müssen, die zu unterschiedlichen Zeitpunkten und mit unterschiedlichen Positionen des Patienten im Gerät erstellt wurden. Aber auch bei Verwendung verschiedener Modalitäten z.B. bei der Registrierung einer MRT-Serie zu einer PET-Serie oder intraoperativem Ultraschall kommen Methoden der Registrierung zum Einsatz. Beim Einsatz verschiedener Modalitäten sind oft komplexere Verfahren nötig, als die, die bei mono-modalen Registrierungen verwendet werden können. Im Rahmen dieser Diplomarbeit müssen verschiedene Ultraschallserien gegeneinander registriert werden, die unter unterschiedlichen Aufnahmewinkeln erstellt wurden.

Viele gängige Algorithmen arbeiten mittels Landmarken. Es werden externe und anatomische Landmarken unterschieden. Anatomische Landmarken können automatisch gefunden oder von Hand in den Bildserien markiert werden. Es werden hierzu möglichst kleine Strukturen verwendet, die in beiden Serien gut erkennbar sind und konstante Position im Vergleich zu anderen Inhalten in den Serien haben. Externe Landmarken können z.B. über am Körper angebrachte Markierungen realisiert werden, die mittels eines Trackingsystems erfasst werden können.

Die Aufgabe des Registrierungsalgorithmus ist es eine Transformation zu finden, die beide Serien möglichst optimal aufeinander abbildet. Dies kann bei der hier verwendeten starren Registrierung z.B. über χ^2-Minimierung erfolgen.

Ein anderer Ansatz ist der Iterative Closest Point (ICP) Algorithmus, der es ermöglicht zwei Punktwolken gegeneinander zu registrieren, um damit z.B. eine gemeinsame Oberfläche zu berechnen.

2.4.2. 3D Rekonstruktion

Bei allen Serien aus CTs, oder MRTs handelt es sich bereits um dreidimensionale Daten. Diese Daten sind bei Grauwertbildern in vielen Fällen als ein- oder mehrdimensionales Array organisiert. Außerdem werden sowohl Anzahl der Bildpunkte in X-, Y- und Z-Richtung, als auch der Abstand der Pixelmittelpunkte zu ihren Nachbarn in allen drei Richtungen angegeben.

Aus diesem dreidimensionalen Datensatz können nun entweder mittels multiplanarer Rekonstruktion zweidimensionale Bilder in beliebigen Schnittebenen erzeugt oder mittels einer dreidimensionalen Rekonstruktion 3D-Ansichten der abgebildeten Objekte erzeugt werden.

Die multiplanare Rekonstruktion umfasst folgende Arbeitsschritte:

- Festlegung von Orientierung der Schnittebene und damit Definition eines Koordinatensystems.
- Festlegen der Auflösung der Ebene
- Nun wird jeder Bildpunkt der neuen Schnittebene in das Koordinatensystem der Serie transformiert und das Array neu abgetastet. Die Grauwerte für den jeweiligen Bildpunkt werden mittels Interpolationsmethoden zwischen den abgetasteten Bildpunkten in der Serie berechnet.

Im klinischen Einsatz sind vor allem die drei Standardorientierungen axial, sagittal und coronal von Bedeutung.

Für dreidimensionale Darstellungen der Serie werden zwei verschiede Techniken verwendet:

- Oberflächenvisualisierung
- Volumenrendering

Bei der Oberflächenvisualisierung wird zuerst ein Schwellwert definiert, der später für die Berechnung der Oberfläche verwendet wird. Die Oberfläche selbst wird durch Triangulation also der Definition von Dreiecken zwischen den Punkten der Oberfläche berechnet. Hierzu werden in den meisten Fällen entweder der Marching Cube Algorithmus oder die Delauney Triangulation verwendet. Beiden gemeinsam ist, dass sie eine für gegebenen Schwellwert vollständig über Dreiecke definierte Oberfläche des Objektes berechnen, die dann geeignet schattiert und dargestellt wird. Der algorithmische Aufwand ist im Gegensatz zum Volumenrendering sehr hoch.

Volumenrendering ist deutlich rechenintensiver und in den meisten Fällen robuster, als Oberflächenvisualisierung. Für das Volumenrendering wird direkt der dreidimensionale Datensatz verwendet. Volumenrendering basiert auf Strahlen ausgehend vom Betrachtungspunkt (View Reference Point). Diese Strahlen sind auf eine Bildebene gerichtet. Betrachtungspunkt und Bildebene werden in das Koordinatensystem der Serie transformiert. Anschließend befindet sich zwischen Betrachtungspunkt und Bildebene die Bildserie. Für jeden einzelnen Strahl durch die Serie werden die Intensitätswerte entlang des Strahls berechnet. Je nach gewünschtem Effekt können sowohl Oberflächen an gegebenen Schwellwerten, sowie Maximal-, oder Minimalwerte entlang des Strahles visualisiert werden.Wegen der Rechenintensität ist Volumenrendering für große Datensätze nur auf schnellen Grafikkarten in Echtzeit durchführbar.

Während dieser Arbeit kam ausschließlich Volumenrendering zum Einsatz, da dieses Verfahren keinerlei Bedingungen an die Bilderie stellt.

Abbildung 10 zeigt im oberen Teil eine volumengerenderte Oberfläche eines Phantoms und im unteren Teil zweidimensionale Slices, die mittels multiplanarer Rekonstruktion erstellt wurden.

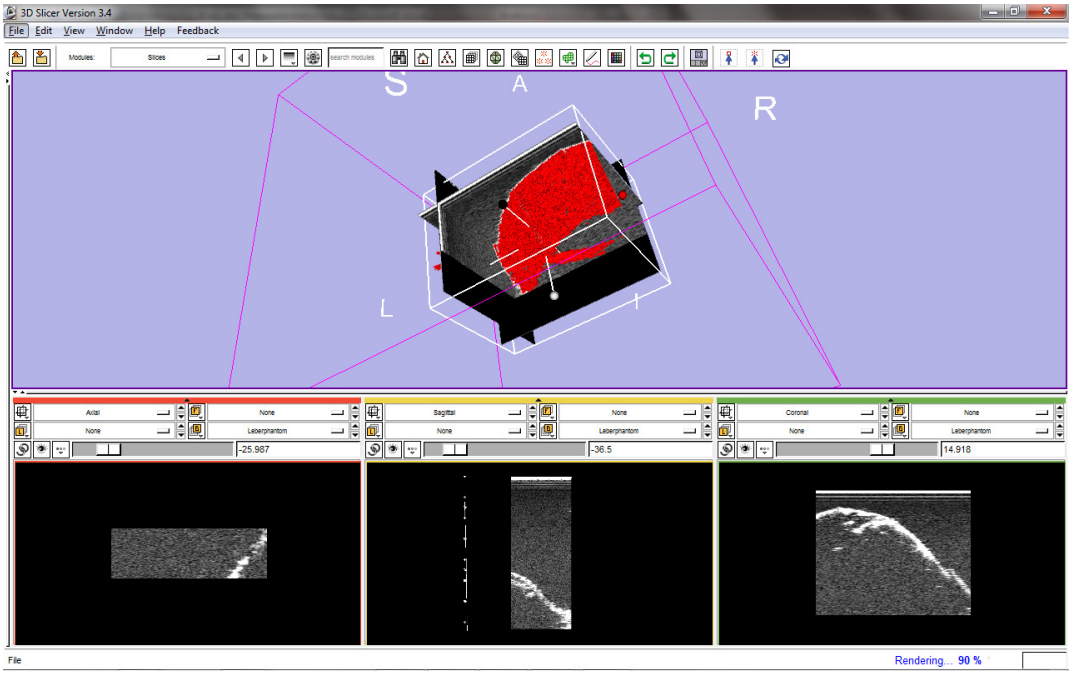

Abbildung 10: volumengerenderte Darstellung der Oberfläche eines Phantoms und multiplanare Rekonstruktion

2.5. Tomographie

Als Tomographie (griechisch: tome-Schnitt, graphein-schreiben) werden schnittbild-erzeugendende bildgebende Verfahren bezeichnet, die in der Medizin von größter Bedeutung sind.

Es ist mittels verschiedener Verfahren möglich, die Anatomie eines Menschen und somit auch pathologische Abweichungen vom Normzustand nichtinvasiv in großer Auflösung sichtbar zu machen. Ebenfalls möglich ist die Darstellung der Funktion von Organen.

Die am häufigsten eingesetzten Verfahren sind Computertomographie und Magnetresonanz-tomographie.

Die Computertomographie nutzt die Gesetzmäßigkeit der Abschwächung der Röntgenstrahlung im Gewebe aus. Der Patient wird von einer um ihn rotierenden Röntgenröhre bestrahlt. Die abgeschwächte Strahlung wird auf der gegenüberliegenden Seite mittels digitaler Detektoren nach dem in Abbildung 11 ersichtlichen Fächerstrahlprinzip aufgezeichnet. Je nach Winkel der Röhre bezüglich der Längsachse des Patienten werden unterschiedliche Detektoren im Ring getroffen.

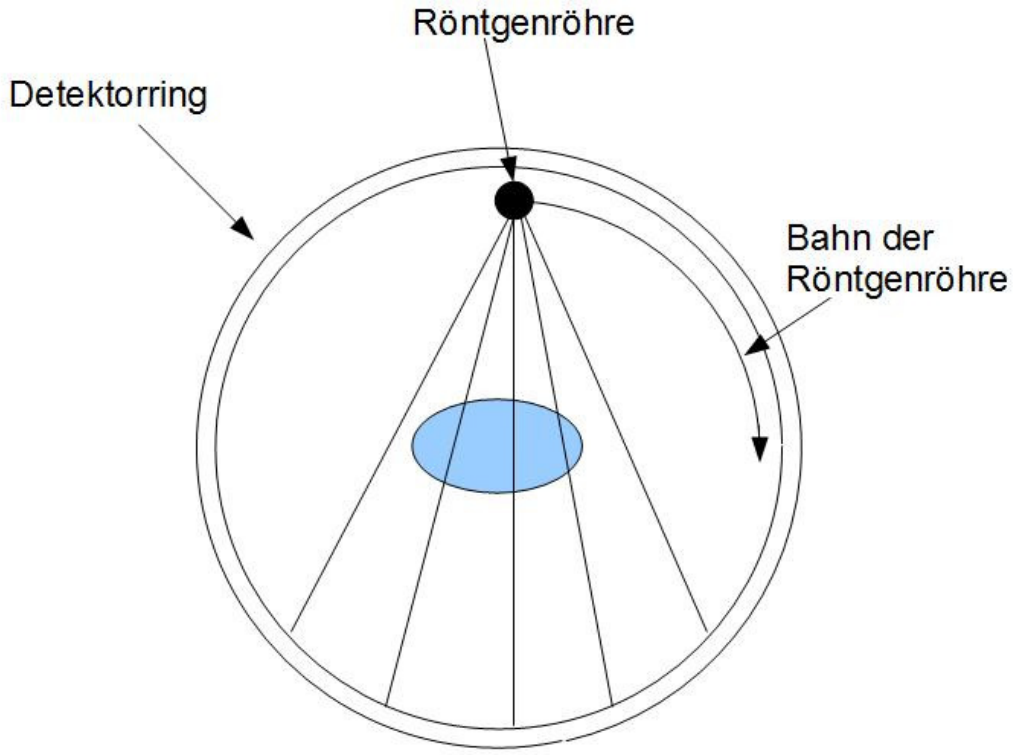

Abbildung 11: Fächerstrahlprinzip bei Verwendung eines Computertomographen

Mittels gefilterter Radon-Rücktransformation kann aus den an den Detektoren gemessenen Strahlungsintensitäten unter Berücksichtigung der Winkel der Röntgenröhre ein Grauwertbild der Anatomie in der bestrahlten Ebene des Patienten rekonstruiert werden.

Wird dieser Vorgang in äquidistanten Abständen entlang des Patienten (axiale Schnitte) wiederholt, entstehen zweidimensionale Schnittbilder des Patienten mit konstanten Abständen. Moderne Geräte sind in der Lage mehrere Ebenen gleichzeitig aufzuzeichnen.

Computertomographen weisen Ortsauflösungen von circa 0.5 mm auf. Die Kontrastauflösung ist für Knochen deutlich höher, als dies bei Magnetresonanztomographen der Fall ist, die aber wiederrum eine deutlich bessere Weichteilauflösung liefern. Außerdem ist der Patient aufgrund der Verwendung von Röntgenstrahlen bei Computertomographie hohen Strahlendosen ausgesetzt, die das Krebsrisiko erhöhen.

Abbildung 12 zeigt einen modernen Computertomographen der Firma Siemens.

Abbildung 12: Siemens Somatom Definition Flash[7]

Bei der Magnetresonanztomographie wird die Präzedenz atomarer Kreisel ausgenutzt. Viele Atomkerne besitzen einen Eigendrehimpuls und damit ein Magnetfeld. Werden nun mittels eines starken Magnetfeldes die Atome aus ausgerichtet, entsteht eine Präzessionsbewegung, die aus der Impuls-erhaltung folgt. Diese arbeitet gegen die Auslenkung der Atome durch das äußere Magnetfeld an und bewirkt, dass die Achse des lokalen Magnetfeldes des Atoms in einer kreisförmigen Bahn um den Feldvektor des Hauptmagnetfeldes mit einer materialspezifischen Larmorfrequenz rotiert. Mittels eines von außen einwirkendenden Hochfrequenzfeldes lässt sich die Richtung der Spins beeinflussen. Werden nun elektromagnetische Wellen mit Larmorfrequenz auf die im Magnetfeld ausgerichteten Atome ausgestrahlt, so werden diese im Vergleich zu ihrer Gleichgewichtslage ausgelenkt. Anschließend wird über eine Leiterschleife die magnetische Induktion durch das lokale Magnetfeld der Atome und damit die Zeit bis zur Rückkehr der Atome zur Gleichgewichtslage gemessen (Induktion nimmt bei Rückkehr zum Gleichgewichtszustand ab). Diese Zeit ist gewebespezifisch und wird zur Bildgebung verwendet.

Die Magnetresonanztomographie liefert ebenfalls Ortsauflösungen im Bereich von 0.5 mm.

Zur funktionalen Bildgebung werden außerdem funktionale Magnetresonanztomographie (fMRT), Single-Photon-Emission-Tomographie (SPECT) und Positronen-Emissions-Tomographie (PET) verwendet.

Auch bei konventionellen B-Mode Ultraschallbildern handelt es sich um tomographische Aufnahmen.

[7] http://www.medical.siemens.com/webapp/wcs/stores/servlet/ProductDisplay~q_catalogId~e_-3~a_catTree~e_100010,1007660,12752,1008408~a_langId~e_-3~a_productId~e_187741~a_storeId~e_10001.htm (letzter Zugriff: 01.12.2010)

3. Stand der Forschung

Kapitel 3 beschäftigt sich mit Entwicklungen in Klinik und Forschung, die für diese Arbeit relevant sind.

Es werden jeweils kurz aktuelle Projekte in 2D- und 3D-Ultraschallbildgebung, Anwendung von Robotik in der Medizin, Ultraschalltomographie, sowie Echtzeitanwendungen in der Medizin über Netzwerkprotokolle beleuchtet.

3.1. 3D-Ultraschall

Dreidimensionale Ultraschallbildgebung wird im klinischen Alltag bereits eingesetzt. So werden dreidimensionale Ultraschallbilder während der intraoperativen Verlaufskontrolle und ganz besonders bei der Schwangerschaftsdiagnostik verwendet.

Besonders häufig werden Verfahren eingesetzt, die das in Kapitel 2.1 beschriebene Speckle-Rauschen ausnutzen. Dieses Speckle Rauschen ist materialspezifisch und entsteht durch Reflexion an Objekten die kleiner sind als das kleinste aufzulösende Objekt der Ultraschallsonde. Die Summe der Reflexionen kann als Speckle-Rauschen gemessen werden. Da die meisten Objekte mehr oder weniger ortsfest sind, kann das Rauschen bei Bewegung der Sonde verfolgt werden und so die Position der Sonde relativ zu vorhergehenden Positionen bestimmt werden. Dieses Verfahren bietet von allen Methoden dreidimensionale Ultraschallbilder zu erstellen die geringste Genauigkeit, da Weichgewebe sich unter Druck verschiebt und sich die Speckles je nach Einfallwinkel des Schalles verschieben können.

Genauer als Verfahren die Speckle Dekorellation verwenden sind extern getrackte Ultraschallsonden. Auch die verwendeten Systeme unterscheiden sich hinsichtlich ihrer Genauigkeit. Verwendet werden elektromagnetische, optische oder mechanische Trackingsysteme. Bauartbedingt liefern mechanische Systeme, bei denen die Sonde starr mit einem Messarm verbunden ist, die höchste Genauigkeit, haben aber Nachteile hinsichtlich Bewegungsfreiheit und Aktionsradius.

Elektromagnetische Trackingsysteme haben wie auch optische Trackingsysteme keine mechanische Verbindung zur Sonde. Elektromagnetische Trackingsysteme verfolgen elektromagnetische Marker, die an bekannten Positionen an der Sonde angebracht sind und bestimmen so Position und Orientierung der Sonde. Elektromagnetische Trackingsysteme haben im Vergleich zu optischen Trackingsystemen, die optische Marker verfolgen, unter Optimalbedingungen häufig eine geringere Genauigkeit, benötigen aber im Vergleich zu diesen keine Sichtverbindung und sind nicht abhängig von Umgebungsbedingungen.

Abbildung 13 zeigt ein optisches zwei Kamera-Trackingsystem der Firma NDI.

Abbildung 13: NDI Polaris Spectra[8]

Von allen Systemen die beste Genauigkeit bieten zweidimensionale Array-Schallköpfe. Diese können aufgrund ihrer Geometrie bereits dreidimensionale Gewebeaufnahmen erstellen, ohne dass diese nachträglich aufwendig verarbeitet werden müssen. 2D-Array Schallköpfe sind unabhängig von Umgebungsbedingungen, haben aber bauartbedingt ein sehr geringes Gesichtsfeld.

All diese Verfahren sind wie bereits erwähnt zwar schon in der Klinik verfügbar, aber dennoch Gegenstand der Forschung. So ist es weiterhin schwierig mittels Speckle-Dekorrelation quantitativ beurteilbare 3D Ultraschallbilder zu erstellen[9]. Deshalb kommt in der Schwangerschaftsdiagnostik in jedem Fall weiterhin konventioneller B-Mode Ultraschall zum Einsatz. Wie bereits in Kapitel 2.1. erwähnt wird bei der Ultraschallbildgebung von einer Schallgeschwindigkeit von 1540m/s im Gewebe ausgegangen. Die Schallgeschwindigkeit in verschiedenen Geweben variiert massiv. So bewegt sich Schall im Muskel mit einer anderen Geschwindigkeit als in Fett. Daraus resultieren geometrische Verzerrungen. Die Vermeidung bzw. Minimierung dieser Verzerrungen ist ein sehr wichtiges Forschungsthema, um nicht nur 3D-, sondern auch 2D-Ultraschallbilder hinsichtlich der Möglichkeit sie quantitativ zu beurteilen zu verbessern[10].

Schließlich wird an Verfahren gearbeitet, die automatische Segmentierung von Objekten auf Ultraschallbildern[11], schnellere Rekonstruktion von 3D Aufnahmen[12] oder Entfernung des Speckle

[8] http://www.ndigital.com/medical/polarisfamily-benefits.php (letzter Zugriff: 01.12.2010)
[9] O. Solberg et al. : Freehand 3D Ultrasound Reconstruction Algorithms – A Review. Ultrasound in Med. & Biol., Vol. 33, No. 7, pp. 991–1009, 2007
[10] D. Napolitano.: Sound speed correction in ultrasound imaging. Ultrasonics 44 (2006) e43–e46
[11] B. Chiu et al.: Three-Dimensional Carotid Ultrasound Segmentation Variability Dependence on Signal Difference And Boundary Orientation. Ultrasound in Med. & Biol., Vol. 36, No. 1, pp. 95–110, 2010
[12] T. R. Nelson et al.: Three Dimensional Ultrasound Imaging. Ultrasound in Med. & Biol., Vol. 24, No. 9, pp. 1243–1270, 1998

Rauschens [13] (z.B. mit Hilfe von waveletbasierten Verfahren) ermöglichen, um die Handhabung und Beurteilung von Ultraschallbildern zu vereinfachen.

3.2. Robotik in der Medizin

Auch die Verwendung von Robotern in der Medizin ist inzwischen im klinischen Alltag angekommen. Allen Systemen voran ist das Da Vinci[14] Robotersystem der Firma Intuitive Surgical zu nennen[15].

Beim Da Vinci Roboter handelt es sich um ein Teleoperationssystem zur Verwendung bei minimalinvasiven Eingriffen vor allem im Bereich der Urologie (Prostatektomie). Da Vinci wird aber auch im Bereich der Herzchirurgie verwendet und ermöglicht hier völlig neue Operationsansätze, ohne den Brustkorb des Patienten eröffnen zu müssen. Das System ist mit einem vier-armigen Operationsroboter ausgestattet, welcher sowohl die Einmalwerkzeuge für den Eingriff, als auch ein Endoskopsystem trägt. Dieses vermittelt dem Operateur einen dreidimensionalen visuellen Eindruck mit bis zu 10-facher Vergrößerung aus dem Inneren des Patienten. Der Operateur sitzt zur Bedienung des Systems an einem Arbeitsplatz, der mit zwei Displays zur Vermittlung des dreidimensionalen Eindrucks sowie zwei vielachsiger Eingabegeräte zur Steuerung der Roboterarme ausgestattet ist. Endoskope und Werkzeuge werden durch sehr kleine Zugangsöffnungen (minimalinvasiv) in den Patienten eingeführt. Die Werkzeuge, die für die Operation notwendig sind, werden durch Assistenten am Operationstisch montiert und demontiert. Intuitive Surgical ist mit diesem System mit über 600 allein in den USA verkauften Einheiten weltweit führend. Da Vinci ist mechanisch sehr aufwändig und bis heute nicht in der Lage, ein haptisches Feedback aus dem Inneren des Patienten an den Chirurgen zu vermitteln. Intuitive Surgical plant dieses Feature in den nächsten Jahren nachzurüsten.

Einen modernen Ansatz verfolgt das deutsche Luft- und Raumfahrtzentrums. Bei MiroSurge[16] handelt es sich um einen dreiarmigen Roboter (zwei Werkzeugarme und ein Arm zum Tragen des Stereoendoskops mit jeweils 7 Freiheitsgraden). Die Arme von MiroSurge sind technisch dem in dieser Abreit verwendeten LWR IV sehr ähnlich. Die Roboterarme sind mit Kraftmomentensensorik in jedem Gelenk ausgestattet und können direkt am Operationstisch montiert werden. Die Werkzeuge des MiroSurge Systems sind sterilisierbar, mehrfach verwendbar und mit miniaturisierter Kraftmomentensensorik ausgestattet. Die Kräfte an der Werkzeugspitze können sowohl über Methoden der Augmented Reality im 3D Bild des Endoskops eingeblendet, als auch direkt auf die 6-Achs-Eingabegeräte in den Händen des Chirurgen übertragen werden. MiroSurge befindet sich aktuell in der Entwicklung.

---

[13] S.Sudha et al.: Speckle Noise Reduction in Ultrasound Images
by Wavelet Thresholding based on Weighted
Variance. International Journal of Computer Theory and Engineering, Vol. 1, No. 1, April 2009

[14] http://www.intuitivesurgical.com/products/davinci_si_surgicalsystem/da-vinci-si-surgical-system-features.aspx (letzter Zugriff: 01.12.2010)

[15] http://www.intuitivesurgical.com (letzter Zugriff: 01.12.2010)

[16] http://www.dlr.de/rm/en/desktopdefault.aspx/tabid-3835/6288_read-9047/ (letzter Zugriff: 01.12.2010) und
http://www.dfg.de/download/pdf/dfg_magazin/wissenschaft_oeffentlichkeit/forschung_magazin/german_research_1_10_en.pdf (letzter Zugriff: 01.12.2010)

Roboter finden in der Medizin nicht ausschließlich in der Teleoperation Anwendung. Am Heidelberger Schwerionentherapiezentrum werden Roboter zur exakten Positionierung der Patientenliege vor dem Austritt des Strahlenganges verwendet, um den Tumor möglichst präzise bestrahlen zu können. Ebenso werden dort C-Bögen robotergestützt positioniert, um vor dem Beginn der Bestrahlung Position und Lage des Tumors erneut zu eruieren.

Einen weiteren Ansatz zur Verwendung von Robotern in der Medizin verfolgen A.Lasso et al.[17] an der Queen's University. Mit Hilfe von 3D Slicer und einem Magnetresonanztomographen wird ein transrektaler Roboter verwendet, um Biopsien der Prostata zu entnehmen. Durch Verwendung des Roboters innerhalb des MRT kann so jederzeit die Position des Roboters, als auch die der Nadel verifiziert und somit die Biopsienadel hochgenau an den gewünschten Punkt in der Prostata navigiert werden.

Wie in dieser Arbeit realisiert und im folgenden Kapitel beschrieben, kommen Roboter auch bei der Ultraschalltomographie zum Einsatz.

3.3. Ultraschalltomographie

Obwohl Ultraschalltomographie schon seit den späten 70er Jahren ein Thema in der medizinischen Forschung ist, sind medizinisch zugelassene Geräte erst in den letzten Jahren marktreif geworden.

So bietet die Firma Techniscan Medical Systems[18] seit kurzem einen Ultraschalltomographen zur Verwendung bei der Brustkrebsvorsorge an. Auch Siemens bietet mit dem Acuson S2000 ABVS[19] ein entsprechendes System an. Beiden Geräten gemeinsam ist dass sie sich ausschließlich zur Bildgebung der weiblichen Brust eignen. Beide Geräte verwenden hierzu allerdings unterschiedliche Verfahren. Während der Techniscan Svara mit getrennten Sende- und Empfangssonden arbeitet, die um die Brust im Wasserbad rotieren und nach jeder vollständigen Rotation um den gewünschten Schichtabstand vorgeschoben werden, verwendet der Acuson S2000 eine kombinierte Sonde, die nach dem Echo-Impuls Verfahren arbeitet und motorisiert über die Brust geschoben wird.

Auch in der Forschung werden weiterhin vor allem diese Verfahren verwendet. Während die im Techniscan Svara eingesetzte Time-Of-Flight Technologie bereits Mitte der 1970er Jahre[20] vorgestellt wurde, wurde die Realisierbarkeit von Ultraschalltomographen mit Echo-Impuls Sonden erst in den späten 90er Jahren mit dem Hippokrates Roboter Arm von F. Pierrot et al. (1999)[21] analysiert.

---

[17] A.Lasso et al.: Robot-assisted MRI-guided prostate biopsy using 3D Slicer, NA-MIC Tutorial Contest: Summer 2010

[18] http://www.techniscanmedicalsystems.com/ (letzter Zugriff: 01.12.2010) und http://www.techniscanmedicalsystems.com/index.php?p=Svara%3Csup%3ETM%3C/sup%3E%20Warm%20Bath%20Ultrasound%20%28WBU%3Csup%3ETM%3C/sup%3E%29 (letzter Zugriff: 01.12.2010)

[19] http://www.medical.siemens.com/webapp/wcs/stores/servlet/ProductDisplay~q_catalogId~e_-11~a_catTree~e_100010,1007660,12761,1003803~a_langId~e_-11~a_productId~e_187041~a_storeId~e_10001.htm (letzter Zugriff: 01.12.2010)

[20] G.H.Glover et al.: Computerized Time-Of-Flight Ultrasound Tomography For Breast Examionation. Ultrasound Med. Biol., VoL. 3, pp. 117-127. Pergamon Press, 1977.

[21] F.Pierrot et al.: Hippocrate: a safe robot arm for medical applications with force Feedback. Medical Image Analysis (1999) volume 3, number 3, pp 285–300

Die aktuell höchste Genauigkeit bietet der USCT von H. Gemmeke et. al. (2007)[22], der nach dem Time-Of-Flight Prinzip arbeitet. Das System arbeitet mit 48 Ultraschallsonden, die in zylindrischer Anordnung in mehreren Ebenen um ein Wasserbad positioniert sind und jeweils 32 Empfangs- und acht Sendeelemente enthalten. Zusätzlich lässt sich die Anordnung um das Wasserbad rotieren. Damit ergeben sich 2304 Sende- und 9216 Empfangspositionen. Während eine Sonde sendet, empfangen alle anderen Sonden im Versuchsaufbau die Schallwellen, die von der sendenden Sonde ausgehen. Jede Sonde sendet zur Bildgebung jeweils einmal Wellen aus, die von allen anderen Sonden detektiert werden. Anschließend sendet die nächste Sonde, während die anderen Sonden detektieren. Dies wird solange wiederholt bis jede Sonde Ultraschallwellen emittiert hat. Danach wird der Aufbau rotiert und alle bisherigen Schritte wiederholt. Bei der Messung von Nylonfäden mit 0.15mm Durchmesser konnte die mittlere Dicke der Fäden mit 0.16mm (0.06mm Standardabweichung) sehr genau bestimmt werden. Dieses System hat aufgrund der Anordnung der Sonden und der gewünschten Genauigkeit einen zylindrischen Arbeitsraum mit 18 cm Durchmesser und 20 cm Höhe und ist somit nicht geeignet für die Bildgebung der Schilddrüse oder des menschlichen Torso.

M. Janvier et al. publizierten 2008 eine Arbeit[23], bei der sie einen Industrieroboter der Firma CRS Robotics Corporation mit einem 6-Achs-Kraftmomentensensor ausstatteten, an den eine Ultraschallsonde montiert war, die robotergeführt zweidimensionale B-Mode Bilder erstellte, welche anschließend zu einem 3D-Datensatz kombiniert wurden. Mit Hilfe dieses Systems, das im Impuls-Echo Verfahren arbeitete, wurde die Genauigkeit des Robotersystems evaluiert. Es stellte sich heraus, dass der Roboter je nach Gelenkstellung Positionierungsgenauigkeiten zwischen 0.46 und 0.75 mm erreichte.

Eine weitere Arbeit von E.Boctor et al. aus dem Jahre 2004[24] verwendete den LARS Roboter, der eine Ultraschallsonde führte, um eine Olive in einer Kalbsleber zu detektieren. Auch dieser Roboter verwendete einen Kraftmomentensensor, um den Anpressdruck auf der Leber konstant zu halten. Nach der erfolgten Berechnung des 3D-Datensatzes konnte die Leber mit der beinhalteten Olive in der Software 3D Slicer visualisiert und eine Biopsie geplant werden, die mittels eines zweiten Roboters durchgeführt werden sollte.

[22] H. Gemmecke et al.: 3D ultrasound computer tomography for medical imaging. Nuclear Instruments and Methods in Physics Research A 580 (2007) 1057–1065

[23] M.Janvier et al.: Performance evaluation of a medical robotic 3D-ultrasound imaging system. Medical Image Analysis 12 (2008) 275–290

[24] E. Boctor et al.: A Dual-Armed Robotic System for Intraoperative Ultrasound Guided Hepatic Ablative Therapy:A Prospective Study. Proceedings of the 2004 IEEE International Conference on Robotics & Automation

3.4. Medizinische Echtzeitanwendungen über TCP/IP

Anwendungen über TCP/IP werden in der Medizin häufig benötigt. Ein telemedizinisches System mit landesweitem oder gar weltweitem Zugang für Spezialisten ist gerade für schwach besiedelte Gebiete, in denen kaum Jemand Zugang zu Spezialkliniken hat, von enormem Nutzen.

In Grönland wurde vom Deutschen Krebsforschungszentrum[25] und der Chili GMBH Heidelberg[26] ein teleradiologisches System [27] aufgebaut, das genau auf die dortigen Anforderungen zugeschnitten ist. So können z.B. Datensätze aus radiologischen Geräten via TCP/IP an Spezialisten übermittelt und deren Befund eingeholt werden, ohne dass diese sich beim Patienten vor Ort befinden müssen. Ebenso ist es möglich sich via Videokonferenz mit anderen Spezialisten zu beraten, um zum Beispiel eine Zweitmeinung einzuholen.

Aber auch im Krankenhaus spielen Anwendungen über TCP/IP eine Rolle. So verfügen nahezu alle Kliniken mit angebundener Radiologie über ein PACS (Picture Archiving and Communication System), mit dem es möglich ist, radiologische Befunde an speziellen Workstations zu erstellen, die sich an beliebigen Standorten im Klinikum befinden können. Die Verteilung funktioniert hier über einen zentralen Server, der Bilddaten speichert, verwaltet und verteilt. Der wohl wichtigste Standard in diesem Zusammenhang ist wohl DICOM. DICOM dient zum verteilen von Bilddaten und den damit verbundenen Metadaten in der Medizin.

Bei all diesen Systemen sollen die Daten möglichst in Echtzeit ausgetauscht werden.

Besonders wichtig ist Echtzeitfähigkeit bei Systemen, bei denen der Arzt eine Untersuchung räumlich getrennt vom Patienten durchführt. Hier müssen sowohl Video-, Audio-, als auch haptische Informationen, die von Instrumenten oder Geräten geliefert werden, möglichst in Echtzeit beim Arzt ankommen. Ebenso wichtig ist die Gegenrichtung. Ein Arzt sollte möglichst in der Lage sein die Instrumente am Patienten in Echtzeit zu führen. Echtzeit ist hierbei kein eng gefasster Begriff, sondern muss je nach Anwendung definiert werden. Beim Transfer eines Video- oder Audiostreams sowie haptischer Informationen muss die maximale Latenzzeit sehr viel kürzer sein, als zum Beispiel beim Austausch von Patientenakten, damit dies als Datenübertragung in Echtzeit empfunden wird. Besonders kurze Latenzzeiten werden vor allem in der Automatisierungstechnik benötigt.

Aufgrund des Aufbaus vieler Netzwerke ist es häufig sehr schwer Echtzeitfähigkeit zu erreichen. So bietet TCP als Netzwerkprotokoll zwar Fehlererkennung an und kann somit gewährleisten, dass nur korrekte Daten wirklich ankommen, hat als Tribut an die erwünschte Eigenschaft aber einen deutlich größeren Overhead als nicht fehlerkorrigierende, schlanke Protokolle wie UDP, und benötigt somit mehr Zeit für die Übertragung der Daten.

Ein weiteres Problem ist der bei Verwendung von Ethernet fehlende Determinismus. Es kann nicht garantiert werden, zu welchem Zeitpunkt ein Paket übertragen wird. Grund dafür ist CSMA/CD. Alle Netzwerkgeräte lauschen auf dem Netzwerk ob die Leitung gerade belegt ist. Ist dies nicht der Fall und hat eines oder mehrere Geräte auf dem Bus Daten zu senden, so wird dies ausgeführt. Allerdings

[25] http://www.dkfz.de (letzter Zugriff: 01.12.2010)
[26] http://www.chili-radiology.com/de/ (letzter Zugriff: 01.12.2010)
[27] U. Engelmann et al.: Das landesweite Teleradiologiekonzept für Grönland. Telemedizinführer Deutschland, Ausgabe 2007

sendet irgendeines der Geräte seine Pakete über das Protokoll. Wann genau das gewünschte Gerät sendet ist nicht deterministisch.

P. Arbeille et al. stellten 2003 ein System vor, welches telemanipulierte Ultraschalluntersuchungen über ISDN oder Satellitenverbindungen ermöglicht[28]. Der Arzt steuerte über ein Eingabegerät eine Ultraschallsonde, die an einem Roboterarm montiert war über die Oberfläche des Patienten. Die akquirierten Daten wurden per Audio- und Videostream übertragen. P.Arbeille et al. stellten eine im Mittel um 50% verlängerte Untersuchungsdauer im Vergleich zu konventionellem Ultraschall fest.

Ein ähnliches System entwickelten A. Gonzales et al. die einen Roboter mit Ultraschallsonde, der über pneumatische Muskeln bewegt wurde, auf dem Patienten anbrachten und diesen Roboter über ein haptisches Eingabegerät via einer ISDN Leitung steuerten[29]. Gute Ergebnisse bezüglich haptischen Feedbacks konnten bis 100 ms Übertragungsdauer über die Leitung erreicht werden. Erst ab 250 ms Übertragungsdauer wurde eine Untersuchung deutlich erschwert.

[28] P. Arbeille et al.: Echographic Examination In Isolated Sites Controlle From An Expert Center Using A 2-D Echograph Guided By A Teleoperated Robotic Arm. Ultrasound in Med. & Biol., Vol. 29, No. 7, pp. 993–1000, 2003

[29] A. Gonzales et al.: TER: a system for robotic tele-echography. Lecture Notes in Computer Science

4. Konzeption

Kapitel 4 beschäftigt sich mit der Gesamtkonzeption des Systems. Es beschreibt die verschiedenen Modalitäten, Softwareprodukte und die Zusammenhänge zwischen diesen.

4.1. Systembeschreibung

Ziel ist es die Machbarkeit des automatischen, hochpräzisen Scannens beliebiger Körperregionen mittels Ultraschalltomographie aufzuzeigen und eine Entwicklungsplattform für Ultraschall-tomographie zu schaffen.

Das Konzept sieht einen Ultraschalltomographen im Echo-Impuls Verfahren vor. Hierzu soll eine handelsübliche Linearultraschallsonde mit 7.5 Mhz Nennfrequenz an verschiedene Robotersysteme der Firmen Stäubli und Kuka montiert werden. Es sollen lineare Trajektorien im Wasserbad und kraftgesteuerte Trajektorien auf rigiden bzw. flexiblen Oberflächen gefahren werden. Während dem Fahren dieser Trajektorien soll der Roboter in äquidistanten Schritten gestoppt werden und eine Schichtaufnahme mittels des DiPhAS Ultraschallsystems akquiriert werden. Anschließend soll mittels der Software 3D Slicer, die um im Rahmen dieser Arbeit entstandene Module erweitert wurde, die Konstruktion dreidimensionaler Voxeldatensätze aus den zweidimensionalen Ultraschallbildern erfolgen. Für die Vernetzung der einzelnen Komponenten kommt OpenIGTLink als leichtgewichtiges Netzwerkprotokoll zum Einsatz. Eine Ausnahme bildet die Anbindung des Kuka LWR IV. Diese wurde im Rahmen eines weiteren Projektes am Institut bereits mittels CORBA und dem Fast Research Interface von Kuka realisiert. Zur Synchronisation der einzelnen Geräte, der Berechnung der Roboterpositionen sowie zur Steuerung des Gesamtsystems kommt MATLAB zum Einsatz.

Abbildung 14 zeigt grob die Zusammenhänge zwischen den einzelnen System Komponenten.

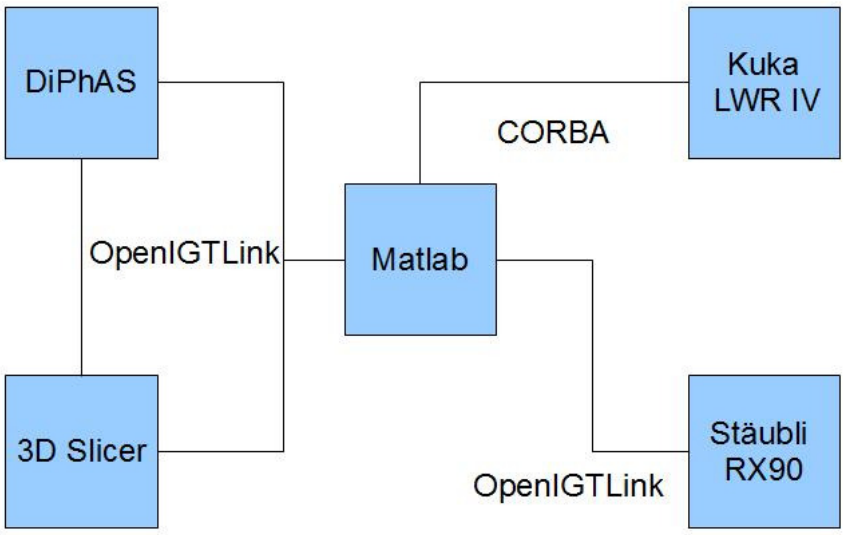

Abbildung 14: Verbindungen zwischen den einzelnen Systemkomponenten

Dieses Konzept ermöglicht eine völlig freie Konfiguration bezüglich einzelner Teile der Anwendung. Es ist möglich beliebige Roboter, Ultraschallgeräte oder Visualisierungskomponenten zu verwenden.

Die Geräte müssen hierzu ausschließlich Status, Image und Transform Nachrichten wie im OpenIGTLink Protokoll definiert empfangen und versenden können. Dies kann zum Beispiel wie im Falle des RX90 durch eine Middleware erfolgen, die OpenIGTLink Nachrichten in für den Roboter verständliche Signale übersetzt.

Die Signalverarbeitung ist so konzipiert, dass 3D Slicer jederzeit Kenntnis über die aktuelle Konfiguration des Ultraschallgerätes hat. Dies ermöglicht eine Adaption der Parameter des Ultraschallgerätes zur Laufzeit der Anwendung und somit eine Optimierung der Scan Parameter.

DiPhAS verwendet Standard Ultraschallsonden. Bei Verwendung des Ultraschalltomographen können alle Linearschallsonden verwendet werden, die auch von DiPhAS unterstützt werden.

Der Aufbau sieht eine Konfiguration des Ultraschallsystems über ein 3D Slicer-Plugin vor. Die Konfiguration des Gerätes selbst wird über eine Software auf Basis des DiPhAS API erfolgen.Nach erfolgter Konfiguration des Ultraschallsystems und Routing zwischen den Systemschnittstellen sollen Initial- und Endpositionen sowie Schrittweite der Roboter in Matlab festgelegt werden.

Der Stäubli RX90 ist mit keinerlei Kraftmomentensensorik ausgestattet und muss somit mittels vom Benutzer explorativ ermittelten Koordinaten gesteuert werden. Für den KUKA LWR IV soll ein Lernmodus implementiert werden, der es ermöglicht Start- und Zielposition durch Bewegen des Roboters in einem kraftgeführten Modus zu ermitteln.

Nach diesem Schritt soll der Benutzer lediglich den Scan mittels eines Knopfdruckes starten. Das System soll nun selbstständig die gewünschte lineare oder kraftgeführte Trajektorie abfahren , die Roboter in äquidistanten Schritten stoppen, Ultraschallaufnahmen akquirieren und diese an 3D Slicer senden, das nach jedem erfassten Bild Matlab mittels einer Statusnachricht den Erhalt signalisiert.

Nach Beendigung des Scans soll ein Slicer-Plugin die Berechnung des 3D-Datensatzes übernehmen. Der entstandene Datensatz soll anschließend in 3D geeignet visualisiert, nachbearbeitet und vermessen werden können.

4.2. OpenIGTLink

Bei OpenIGTLink [30] handelt es sich wie bereits erwähnt um ein leichtgewichtiges Nachrichtenprotokoll. OpenIGTLink wurde für Anwendungen der Image Guided Therapy also für blddatengesteuerte Therapie konzipiert. Für IGSTK existiert eine Anbindung an OpenIGTLink, so dass ohne großen Aufwand fast alle erhältlichen Trackingsysteme verwendet werden können.

OpenIGTLink ist für die Anwendung auf Basis der Protokolle TCP/IP gedacht, ermöglicht aber auch die Verwendung anderer Protokolle wie UDP. Im Rahmen dieser Arbeit wurde OpenIGTLink auf Basis von TCP/IP verwendet.

OpenIGTLink Anwendungen arbeiten in der Regel nach dem Client-Server Prinzip. Eine Modalität stellt hierbei den Server dar und öffnet einen oder mehrere Ports mit dem/denen sich dann jeweils ein Client verbinden kann. Anschließend können Daten ausgetauscht werden.

OpenIGTLink ist mittels des OpenIGTLinkIF Moduls nahezu nahtlos in 3D Slicer ab Version 3.4 eingebunden. Empfangene Nachrichten lösen im Eventsystem von Slicer Ereignisse aus, auf die reagiert werden soll. Objekte, die im OpenIGTLinkIF Modul an einen OpenIGTLink Port gebunden sind, lösen den Versand einer Nachricht aus, sobald ein Ereignis auf ihnen auftritt.

OpenIGTLink Nachrichten sind alle nach demselben Schema aufgebaut. Jede Nachricht besteht aus einem Header und einem Body. Der Header beinhaltet Version des Protokolls, Typ der Nachricht, Name der sendenden Modalität, einen optionalen Timestamp zur Synchronisation, die Größe des Bodies und 64 bit Checksummen Informationen (CRC), um die Korrektheit der empfangenen Daten zu verifizieren.

Der nachfolgende Body variiert je nach Nachrichtentyp

Version 1 des Protokolls beinhaltet folgende Nachrichtentypen:

IMAGE	zur Übertragung von Bildern
TRANSFORM	zur Übertragung von homogenen Koordinaten
POSITION	zur Übertragung von Position als Vektor und Orientierung in Form eines Quaternions
CAPABILITY	zur Übertragung von Möglichkeiten, die ein Gerät bietet (mögliche Kommandos)
STATUS	zur Übertragung von Statusinformationen
GET_CAPABIL	zur Anfrage von Möglichkeiten , die ein Gerät bietet (mögliche Kommandos)
GET_STATUS	zur Anfrage eines Gerätestatus

Version 2 des Protokolls soll Anfang 2011 veröffentlicht werden, kann aber bereits jetzt verwendet werden. Version 2 ist vollständig abwärtskompatibel zu Version 1 und erweitert dieses um Nachrichten zur Abfrage von Sensorik oder zur Übertragung von Arrays, Trajektorien etc.

IMAGE, TRANSFORM und STATUS sollen während dieser Diplomarbeit verwendet werden und werden deshalb im Folgenden näher beleuchtet.

[30] http://www.na-mic.org/Wiki/index.php/OpenIGTLink (letzter Zugriff: 01.12.2010)

Der Body der TRANSFORM Nachricht besteht aus 12 Bytes zur Übertragung der ersten drei Zeilen einer 4x4 Matrix, die homogene Koordinaten enthält. Die letzte Zeile einer homogenisierten Koordinatenmatrix enthält wie in Kapitel 2.3 beschrieben immer die Elemente 0,0,0,1.0 und muss somit nicht übertragen werden.

Der Body der STATUS Nachricht beginnt mit einer unsigned short, die den Code des Status enthält. Anschließend folgt eine 64 bit Integer, die einen SubCode enthält, der den Status Code näher spezifiziert. Außerdem wird ein Array bestehend aus 20 Chars übertragen, um dem Empfänger eine Fehlernachricht zu übermitteln. Schließlich kann optional noch eine beliebig lange Status Nachricht als Characterarray übertragen werden.

Der Body der IMAGE Nachricht enthält einen IMAGE Header, der Position, Orientierung, Auflösung in alle Richtungen (X, Y, Z) sowie Typ der Bilddaten (Skalar, Vektor, Grauwert, RGB) umfasst. Auf den IMAGE Header folgt ein IMAGE Body, der die Bilddaten enthält. Im Header existiert ein Feld Scalar Type, über den definiert wird, welchen Datentyp (int, float, etc.) die im IMAGE Body enthaltenen Bilddaten haben. Die einzelnen Pixelinformationen werden einfach im Sinne eines eindimensionalen Arrays hintereinander in den IMAGE Body geschrieben.

Für jede Nachricht wird vor dem Versand eine Checksumme erstellt, die im Header enthalten ist und auf der Empfangsseite zur Überprüfung der empfangenen Nachricht verwendet wird. Außerdem wird der Body jeder Nachricht vor dem Versand gepackt und nach dem Empfang entpackt, um die Byte Reihenfolge an Modalität bzw. das Netzwerk anzupassen.

Im Rahmen dieser Arbeit wird außerdem noch eine weitere Nachricht zum Austausch der Konfigurationsdaten des Ultraschallgerätes benötigt. Die Implementierungsdetails hierzu sind Kapitel 5 zu entnehmen.

OpenIGTLink stellt eine flexible Möglichkeit dar, Daten mit kurzen Latenzzeiten über das Netzwerk zwischen verschiedenen Modalitäten zu übertragen. Die Integration in Slicer macht OpenIGTLink für diese Arbeit besonders geeignet. Des Weiteren existieren Implementierungen für C++ und Matlab, die ebenfalls für Teile dieser Arbeit in modifizierter Form benötigt werden.

4.3. 3D Slicer

3D Slicer[31] ist eine Open Source Software zur Visualisierung und Bearbeitung von medizinischen Bilddaten. 3D Slicer basiert auf ITK[32] sowie VTK[33] und bietet damit die Möglichkeit, in vollem Umfang auf die Funktionen dieser Toolkits zuzugreifen. So bietet 3D Slicer die Möglichkeit multiplanare Rekonstruktionen durchzuführen und einzelne Slices in einer 3D Ansicht darzustellen. Obwohl 3D Slicer nicht als medizinisch zugelassenes Produkt konzipiert ist, wurde es speziell auf die Bedürfnisse von Ärzten zugeschnitten. 3D Slicer ist deshalb und aufgrund der flexiblen und einfachen Entwicklung von Modulen ein häufig eingesetztes Werkzeug in der interdisziplinären medizinischen Grundlagenforschung geworden. Besonders populär ist Slicer vor allem bei der Arbeit mit Datensätzen, die MRT-, CT- oder Ultraschallgeräten Geräten entstammen.

3D Slicer basiert auf einem leichtgewichtigen Kern, um den die Anwendung mit Modulen aufgebaut ist. 3D Slicer lässt sich hierbei frei über mehrere Wege mit Modulen erweitern.

Die Ansichten von 3D Slicer lassen sich beliebig konfigurieren und flexibel auf dem Bildschirm positionieren. Standardmäßig sind im rechten oberen Anteil des Slicer Fensters die 3D Ansicht und darunter drei zweidimensionale Ansichten zur Anzeige einzelner Slices in den Orientierungen axial, sagittal und coronal geladen. Im linken Teil der Ansicht befindet sich das aktuell geladene Modul und darunter Kontrollmodule zum Manipulieren der 2D- und 3D-Ansichten. Die Kontrollleiste am oberen Bildschirmrand ermöglicht es, Module zu laden, anzuzeigen und zu konfigurieren.

Abbildung 15 zeigt die Oberfläche von 3D Slicer in Standardkonfiguration mit geladenem „Volumes“ Modul.

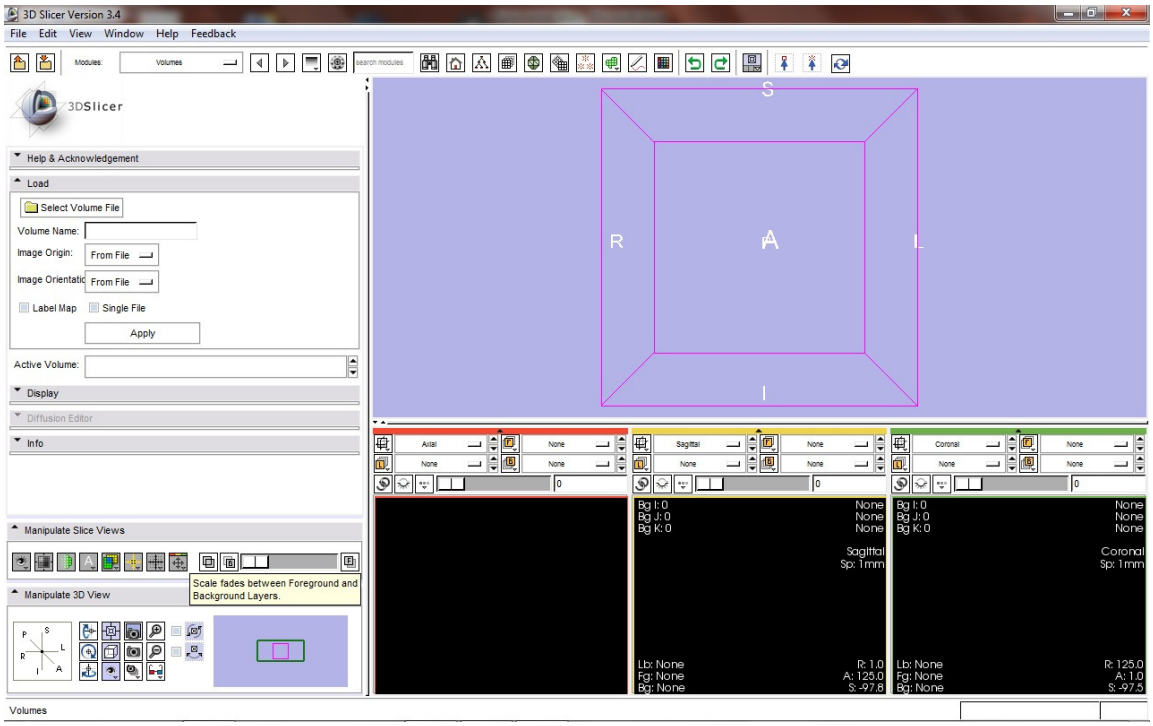

Abbildung 15: Oberfläche von 3D Slicer

[31] http://www.slicer.org/ (letzter Zugriff: 01.12.2010)
[32] http://www.itk.org/ (letzter Zugriff: 01.12.2010)
[33] http://www.vtk.org/ (letzter Zugriff: 01.12.2010)

3D Slicer ist zum Zeitpunkt der Verfassung dieser Arbeit in Version 3.6 verfügbar. Alle Versionen von 3D Slicer 3 verwenden als GUI Toolkit KWWidgets. 3D Slicer 4 befindet sich aktuell in Entwicklung. Module die für Version 3 entwickelt wurden, müssen angepasst werden, da Version 4 als GUI Toolkit aus Performancegründen QT verwendet und die Kompatibilität zu KWWidgets eingestellt wird.

Aufgrund des sehr frühen Alpha-Stadiums von 3D Slicer 4 soll in dieser Arbeit 3D Slicer 3.6 verwendet werden.

3D Slicer ist als plattformunabhängiges Entwicklungswerkzeug entwickelt worden. Als plattformunabhängigs Build System kommt CMAKE zum Einsatz, welches auch bei OpenIGTLink verwendet wurde.

CMAKE erstellt aus vorliegenden Quellcodedateien und einer CMAKE-spezifischen Konfigurationsdatei nach Ermittlung der Umgebung (Compiler und Betriebssystem) plattformspezifische Makefiles, mit denen die Projekte danach kompiliert werden können. Dies befreit den Anwender bzw. Entwickler von aufwändigen Konfigurationen auf dem gewünschten Zielsystem. CMAKE ist außerdem in der Lage, benötigte Bibliotheken auf dem Zielsystem zu lokalisieren und in das Makefile einzubinden.

3D Slicer wurde zu großen Teilen in C++ entwickelt. Um eine größere Flexibilität vor allem bei der Generierung der GUI oder der Integration von Modulen zu erreichen, kommt die Skriptsprache TCL/TK zum Einsatz. So können bei der Generierung einer GUI die Oberflächenobjekte wie Buttons oder Input Felder in C++ erstellt werden und mittels TCL/TK auf der Oberfläche ähnlich der Verwendung von CSS bei HTML positioniert werden.

Module und deren Einstiegspunkte werden ebenfalls automatisiert über TCL/TK Skripte gefunden und eingebunden.

Enorme Vorteile bietet TCL/TK auch bei der Installation von 3D Slicer. Zur Entwicklung auf Basis von 3D Slicer ist aufgrund des leichtgewichtigen Kerns und der losen Kopplung der Komponenten kaum tiefergehendes Architekturwissen von Nöten. Zur Installation müssen allerdings alle notwendigen Bibliotheken heruntergelanden, richtig konfiguriert und installiert werden. Das mit 3D Slicer mitgelieferte Skript getbuildtest.tcl lädt diese Bibliotheken und 3D Slicer herunter, übernimmt die Konfiguration in CMAKE, kompiliert alle Bibliotheken sowie 3D Slicer und führt anschließend Systemtests durch. Dies bietet Anwendern und Entwicklern einen deutlich erleichterten Einstieg in die Verwendung von 3D Slicer. Außerdem bietet getbuildtest.tcl die Möglichkeit, einzelne oder auch alle Komponenten der Anwendung automatisch zu updaten.

Mit dem aktuellen Release wird eine Reihe von nützlichen Modulen mitgeliefert. So sind Module zum Laden von Modulen, Bearbeiten von Transformationen, Volumenrendering über die VTK Pipeline, Filtern, Messen, Registrieren, Segmentieren oder aber zum Setzen von Landmarken oder zur Kontaktaufnahme zur Außenwelt (z.B. über das bereits erwähnte OpenIGTLinkIF Modul) enthalten.

Wegen des großen Nutzerstammes und der Community hinter 3D Slicer existieren aber auch Module für weniger häufig gebrauchte Anwendungen wie Neuronavigation oder Diffusion Tensor Imaging.

Anwendungsfälle in 3D Slicer sind als MRML[34] Szenen definiert. Beim Start von 3D Slicer wird eine leere MRML Szene geladen. MRML ist die Multimedia Retrieval Markup Language und bildet den Kern eines Anwendungsfalles. MRML ist ein XML-basiertes Kommunationsprotokoll. Alle Objekte, seien es Transformationen, Volumen, Landmarken oder Netzwerkverbindungen, werden im „MRML Tree“ von Slicer als MRML Objekte abgelegt. Der MRML Tree bildet hierbei die aktuell geladene Szene. Slicer 3D ist in der Lage, Teile der Szene in anderen Formaten als MRML zu speichern und zu laden. So bietet es sich aus Kompatibilitätsgründen an, Volumen als VTK oder NRRD Dateien zu speichern.

Des Weiteren sind große Teile der Anwendung eventgesteuert. Module können auch aktiv sein, wenn sie aktuell nicht angezeigt werden. Sie müssen ausschließlich beim Start von 3D Slicer geladen werden. Wird nun ein Objekt aus dem MRML Baum in einem Modul verwendet und die Bindung nicht wieder gelöst, lauscht die Anwendung nach Events auf den gebunden Objekten. Bei einem aufgetretenen Event in einem gebundenen Objekt im MRML Baum wird im entsprechenden Modul die Methode ProcessMRMLEvents aufgerufen. Hier kann dann modulspezifisch auf dieses Event reagiert werden. Analog wird bei GUI Aktionen im zum GUI gehörenden Modul die Methode ProcessGUIEvents aufgerufen. Ein Event wird zum Beispiel bei der Änderung eines Transformationsknotens oder dem Drücken eines Buttons ausgelöst. Objekte können selbstverständlich auch in mehreren Modulen zu gleich observiert werden.

Soll ein Modul für 3D Slicer entwickelt werden, so muss je nach Anwendungszweck eines der verfügbaren Modulkonzepte gewählt werden. 3D Slicer bietet aktuell drei verschiedene Möglichkeiten, Module zu entwickeln[35].

1. Command Line Modules
   Command Line Modules bieten eine einfache Möglichkeit eigene Erweiterungen für Slicer zu entwickeln. Speziell für diesen Modultyp wurde das Slicer Execution Model entwickelt.
   Command Line Modules empfehlen sich dann, wenn kein direkter Zugriff auf das Slicer API benötigt wird. 3D Slicer kann Module laden, die als ausführbare Dateien, Shared Objects oder Python Skript Plugins vorliegen. Die Module müssen hierzu eine Beschreibung als XML Datei enthalten und definierte Einstiegspunkte beinhalten. Der Datenaustausch zwischen Slicer und dem Modul erfolgt über Kommandozeilenparameter, die durch den Slicer Command Line Parser verwaltet werden. Mittels des Slicer Execution Model ist es möglich, auch ohne die direkte Integration des Moduls in die Hauptanwendung eine GUI für das Modul in dieser zu erstellen. Hierzu ist dem Modul eine XML Datei beizufügen, die beim Laden des Modules geparst wird. Aus den Informationen in dieser Datei kann Slicer mittels TCL/TK Skripting eine GUI erstellen, die mit dem Command Line Modul interagiert. Command Line Modules sind nicht direkt in 3D Slicer integriert, sondern laufen eigenständig und werden über die Kommandozeilenparameter gesteuert.

[34] http://www.mrml.net/ (letzter Zugriff: 01.12.2010)
[35] http://www.slicer.org/slicerWiki/index.php/Slicer3:Extensions (letzter Zugriff: 01.12.2010) und http://www.slicer.org/slicerWiki/images/7/75/Integrating_with_Slicer3.ppt (letzter Zugriff: 01.12.2010)

2. Skripted Modules
   Skripted Modules bieten die Möglichkeit, Erweiterungen für Slicer direkt in TCL/TK oder Python zu schreiben. Auch bei Skripted Modules bietet Slicer die Möglichkeit, eine GUI in der Hauptanwendung zu generieren. Skripted Modules bieten sich zum Beispiel für die Kommunikation mit externen Anwendungen an, die ebenfalls Python Skripting unterstützen.

3. Loadable Modules
   Loadable Modules bieten vollen Zugriff auf das Slicer API und alle Teile der 3D Slicer Anwendung. Aufgrund der geplanten engen Integration des Ultraschalltomographen in 3D Slicer sollen in dieser Arbeit Loadable Modules verwendet werden.
   Loadable Modules werden ebenfalls über CMAKE konfiguriert. Die Konfigurationsdatei enhält hierzu eine Zeile zum generieren des Loadable Modules (generateLM). GenerateLM aktiviert die Unterstützung für Loadable Modules für das entsprechende Modul. GenerateLM erhält als Parameter Referenzen auf die Quellen des Modules sowie die Modulbeschreibung, die in Form einer txt oder xml Datei vorliegen kann.
   Loadable Modules benötigen eine GUI Klasse, in der über eine Init Methode das Modul und die GUI initialisiert werden. Die Init Methode wird hierbei beim Laden des Moduls also beim Start von Slicer aufgerufen. Sollen Initialisierungen beim Öffnen des Moduls vorgenommen werden, ist außerdem die Enter Methode zu implementieren. Um z.B. das Modul zu deaktivieren, wenn ein anderes Modul geöffnet wird, kann die Exit Methode verwendet werden.
   Jedes Modul muss außerdem eine Logic Klasse haben, die der GUI Klasse bekannt ist. So können Aktionen auf der GUI nach dem MFC Prinzip an die Logik weitergeleitet werden, um hier Manipulationen auf MRML Objekten durchzuführen oder Signale an die Außenwelt zu leiten. Über die bereits erwähnten Methoden ProcessMRMLEvents und ProcessGUIEvents, die sowohl in GUI als auch Logic implementiert werden können, können GUI und Logic getrennt voneinander auf Events reagieren.
   Außerdem können noch anwendungsspezifische MRML Knoten erstellt werden, die nach Laden eines Modules und Bekanntmachung der Modulquellen bei anderen Modulen von diesen mitverwendet werden können.

Abbildung 16 zeigt die Architektur der Anwendung 3D Slicer, die um den MRML Kern aufgebaut ist.

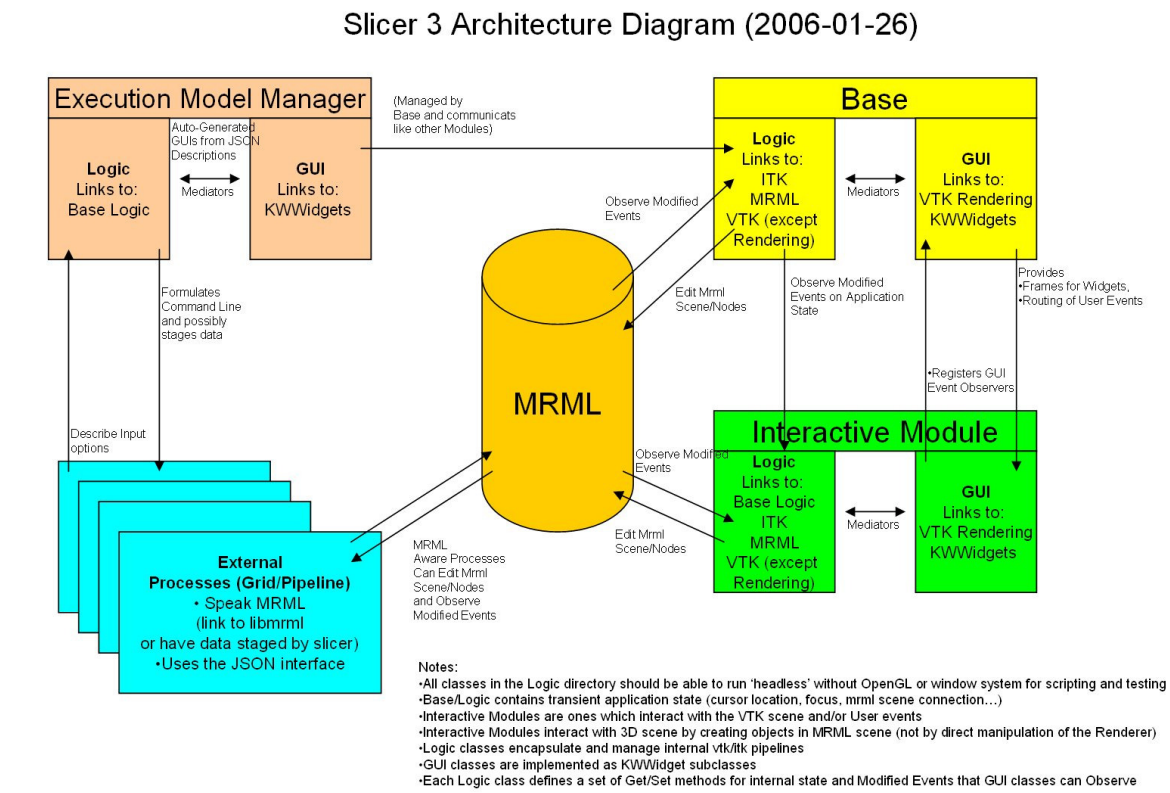

Abbildung 16: Architektur der Anwendung 3D Slicer [36]

[36] http://www.slicer.org/slicerWiki/index.php/Slicer3:Developers (letzter Zugriff: 01.12.2010)

4.4. Matlab

Matlab[37] ist eine von Mathworks entwickelte Software zur Lösung mathematischer Probleme. Matlab beinhaltet eine Programmiersprache mit der Programmcode entwickelt werden kann, der im Gegensatz zu Programmcode, der in Hochsprachen wie C++ entwickelt wurde, nicht kompiliert, sondern interpretiert wird. Hierbei erfolgt die Analyse des Quellcodes zur Laufzeit des Programmes. Matlab eignet sich wie auch Computeralgebrasysteme (z.B. Maple) zum Lösen komplexer mathematischer Probleme, verfolgt aber im Gegensatz zu diesen einen numerischen Ansatz. Matlab ist für den Einsatz mit Matrizen konzipiert und arbeitet mit diesen besonders effizient. In Matlab können m-Files entwickelt werden, die als Matlab Skripte bezeichnet werden, Matlab Code enthalten und in der Matlab Softwareumgebung ausgeführt werden können. Des Weiteren beinhaltet Matlab die Software GUIDE, die sich zur Entwicklung von grafischen Oberflächen für Matlab Skripte eignet. Matlab bietet weiterhin umfassende Möglichkeiten Berechnungen, oder Formeln zu visualiseren. Außerdem ist Matlab Basis für SIMULINK, eine grafische Programmiersprache zur Simulation von komplexen Abläufen, wie sie zum Beispiel in der Regelungstechnik zum Einsatz kommen.

Mathworks bietet weiterhin Toolboxen zur Erweiterung von Matlab an, mit denen sich aerodynamische, mechanische, statistische oder Probleme der Bild- und Signalverarbeitung lösen lassen. Mit der Symbolic Math Toolbox wurde vor einiger Zeit das Computeralgebrasystem MUPAD in Matlab integriert. Damit ist es nun möglich, wie in Maple auch in Matlab symbolisch zu rechnen.

In dieser Arbeit soll Matlab für Probleme der linearen Algebra zum Einsatz kommen. Matlab soll hierzu die Berechnungen der Sondenpositionen und der Roboterposition sowie die Steuerung und Synchronisation der Modalitäten übernehmen. Hierzu muss Matlab um Schnittstellen erweitert werden, die es ermöglichen Kontakt zu 3D Slicer, dem DiPhAS Ultraschallsystem und den Robotern aufzunehmen. Für den Zweck der Erweiterung von Matlab durch Fremdcode ist von Mathworks der Matlab Compiler vorgesehen. Mittels dem Matlab Compiler kannn C, C++ oder FORTRAN Code in Matlab und damit auch in Matlab Skripte eingebunden werden. Jede Datei, die mit dem Matlab Compiler kompiliert werden soll, muss mex.h inkludieren. Mex.h enthält Routinen, die als Schnittstelle zwischen Matlab und der betreffenden Programmiersprache fungieren. So können Parameter und Aufrufe zwischen Matlab und C, C++ oder FORTRAN ausgetauscht werden. Für OpenIGTLink existiert eine Opensource Implementierung für Matlab. Die Anforderungen an den Ultraschalltomographen gehen über die Möglichkeiten dieser Implementierung hinaus, weshalb im Rahmen dieser Arbeit Modifikationen daran vorgenommen werden sollen. Während eines anderen Projektes am Institut für Prozessrechentechnik, Robotik und Automation entstanden außerdem Matlab Schnittstellen, die via CORBA Kontakt zum Kuka LWR IV aufnehmen. Diese sollen ebenfalls zur Realisierung des Tomographen verwendet werden.

Konkurrenzprodukt zu Matlab ist GNU/Octave. Hierbei handelt es sich ebenfalls um eine Interpretersprache, die für den Einsatz mit Matrizen ausgelegt ist. GNU/Octave Code ist in weiten Teilen kompatibel zu Matlab Code.

[37] http://www.mathworks.de/ (letzter Zugriff: 01.12.2010)

4.5. DiPhAS

Bei DiPhAS (Digital Phased Array System)[38] handelt es sich um eine Ultraschallforschungsplattform des Fraunhofer Institut für biomedizinische Technik. DiPhAS hat keine medizinische Zulassung, bietet im Vergleich zu den meisten erhältlichen Forschungsplattformen aber mehr Anwendungsmöglichkeiten. So beherrscht DiPhAS sowohl die Ultraschallmodi A, B und M, als auch einen Farbdopplermodus. Eine Besonderheit des DiPhAS Systemes ist es, dass der Anwender Einfluss auf alle Phasen der Signalgenerierung sowie auf alle Phasen der Signalakquisition hat. DiPHAS ist in der Lage, mit den meisten auf dem Markt befindlichen Ultraschallsonden zu arbeiten. Hierzu benötigt DiPhAS eine XML-Konfigurationsdatei, die Informationen über die Betriebsgrenzen der verwendeten Sonde enthält.

Abbildung 17 zeigt das am Institut verwendete DiPhAS System, Abbildung 18 die während dieser Arbeit verwendete Schallsonde.

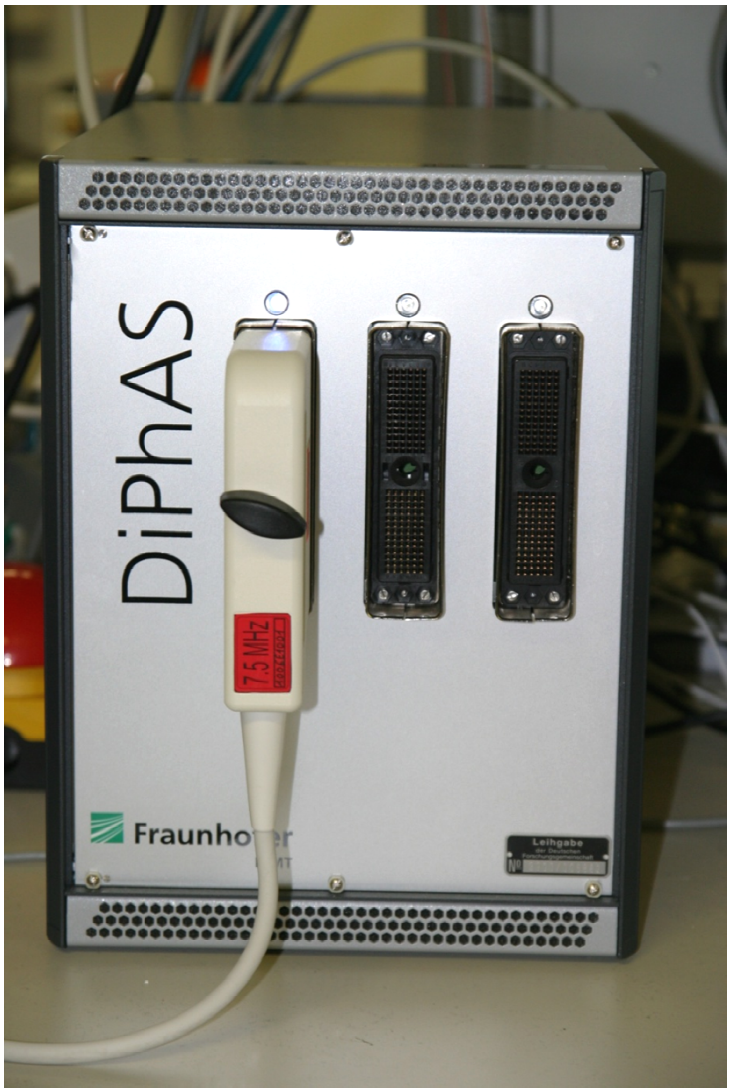

Abbildung 17: DiPhAS Ultraschallsystem

[38] http://www.ibmt.fraunhofer.de/fhg/ibmt/projekte/digitales_phased_array_system.jsp (letzter Zugriff: 01.12.2010)

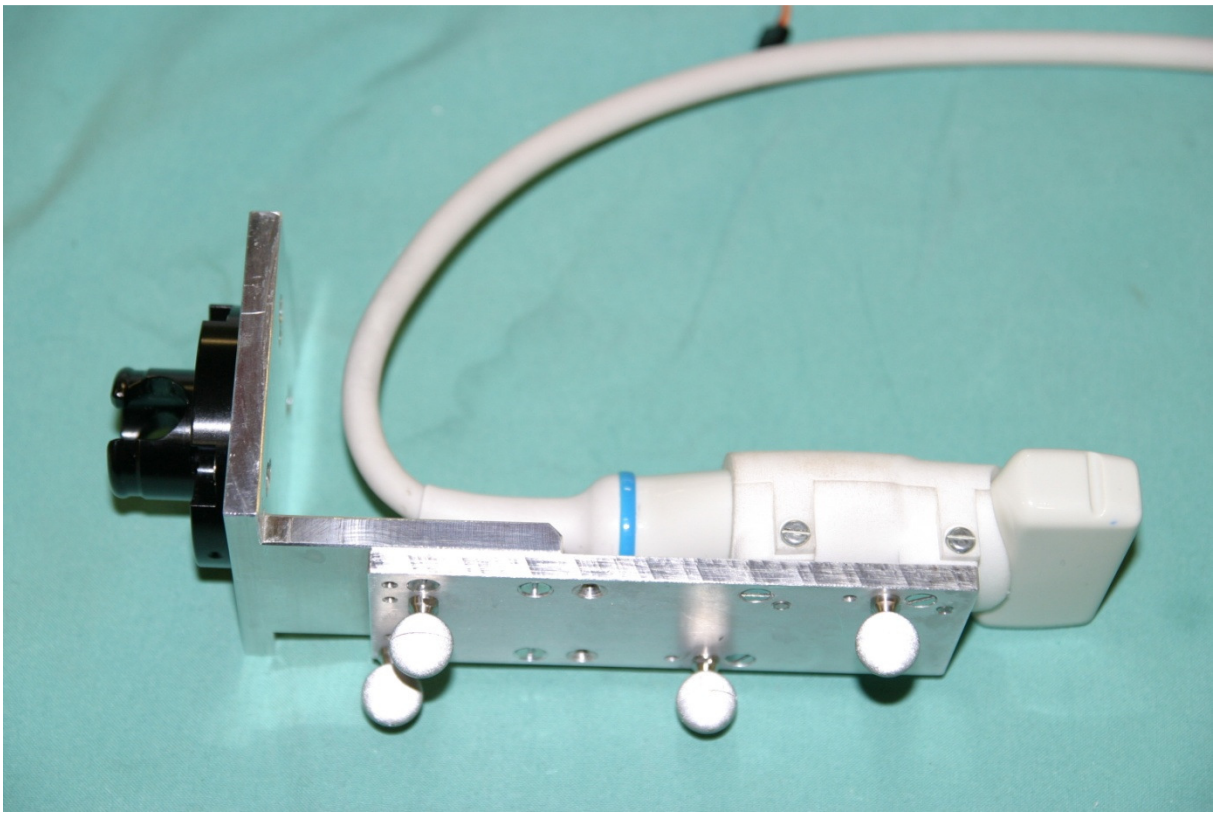

Abbildung 18: 7.5 MHz Linearschallsonde mit montierter Roboterhalterung

DiPhAS besteht aus einer Signalverarbeitungskette, die mittels mehrerer FPGAs (Field Programmable Gate Arrays), die Strahlgenerierung sowie die Signalakquise übernimmt. Die Konfiguration sowie die Übermittlung der empfangenen Signale an den im selben Gehäuse integrierten Quad-Core Windows PC erfolgen über die USB Schnittstelle. Entwickler haben die Möglichkeit, über das in .NET entwickelte DiPhAS API Anwendungen für DiPhAS zu schreiben. Außerdem wird zusammen mit DiPhAS eine auf Basis dieses API entwickelte Anwendung ausliefert, die einen Stand-Alone-Betrieb ermöglicht.

Über das DiPhAS API können alle relevanten Schritte der Signalgenerierung beeinflusst werden. Der Anwender hat dabei Zugriff auf Fokussierung, Wichtung der Sendeamplituden, Kodierung der Sendepulse sowie Ansteuerung von Subapperturen. Außerdem ist es möglich, die Rohdaten einzelner Sende-Empfangselemente zu lesen. Diese Rohdaten (RAW) stellen die unverarbeiteten Daten dar, die von den einzelnen Piezokristallen geliefert werden. Alternativ kann der Benutzer Signale an weiteren Verarbeitungsstufen abgreifen. So ist es ebenfalls möglich, die HF-(Hochfrequenz) Daten einer Arrayuntergruppe zu lesen.

Da das Konzept des Ultraschalltomographen die Verwendung des OpenIGTLink Protokolls für die Konfiguration des DiPhAS und die Übertragung der Bilder vorsieht, ist es notwendig auf Basis des DiPhAS API eine Anwendung zu schreiben, die über empfangene Konfigurationsparameter die Konfiguration des DiPhAS vornimmt. Außerdem müssen die empfangenen B-Mode Daten über eine weitere Schnittstelle an 3D Slicer gesendet werden. Da für .NET keine Implementierung von OpenIGTLink existiert, ist es notwendig einen Wrapper für die C++ Implementierung von OpenIGTLink zu programmieren. Näheres hierzu ist Kapitel 5 zu entnehmen.

4.6. Stäubli RX90

Der Stäubli RX90 [39] ist einer der beiden in der Medizingruppe des IPR verwendeten Robotertypen. Dieser Robotertyp weist eine hohe Positionierungsgenauigkeit auf und ist in der medizinischen Forschung sehr beliebt.

Der RX90 ist ein Sechs-Achs-Knickarmroboter, der mit Positionssensorik in den Gelenken ausgestattet ist. Ohne externen Krakftmomentensensor bietet der RX90 keine Möglichkeiten genaue Kräfte zu messen, die auf die Robotergelenke wirken. Hinsichtlich der Wiederholungsgenauigkeit, die bei vielen anderen handelsüblichen Knickarmrobotern bei circa 0.05 mm liegt, ist der RX90 mit 0.02 mm gegenüber Diesen deutlich genauer. Weiterhin bietet er große Verfahrgeschwindigkeiten und einen nahezu kugelförmigen Arbeitsraum mit 985 mm Radius. Die Nominallast bei circa 100 kg Robotergewicht beträgt 6 kg, während die Maximallast mit 11 kg deutlich höher liegt. Der verwendete Roboter wurde 1999 gebaut und ist auf einem fahrbaren Sockel montiert, so dass er beliebig positioniert werden kann. Das System ist in Abbildung 19 zu sehen.

Abbildung 19: Stäubli RX90 auf dem Sockel

[39] http://www.staeubli.de/ (letzter Zugriff: 01.12.2010) und Stäubli RX90+CS7 Anleitung

Der RX90 ist mit einer in Abbildung 20 sichtbaren externen Steuerung des Typs CS7 ausgestattet. Die Versorgungsspannung beträgt 360 Volt. Die Steuerung kann entweder über einen kabelgebundenen Handcontroller geregelt werden oder über Programme, die mittels eines 3.5“ Diskettenlaufwerks eingespielt werden können. CS7 arbeitet mit einem Echtzeitbetriebssystem (AdeptV+[40])der Firma Adept, das auch eine Programmiersprache für die Roboterprogramme enthält. Des Weiteren kann die Steuerung mit Steckkarten ausgestattet werden, die die Anbindung an Bussysteme oder Ethernet ermöglicht.

Abbildung 20: Stäubli CS7

[40] http://www.adept.de/produkte/software/embedded/v-plus/allgemeines (letzter Zugriff: 01.12.2010)

Der hier verwendete RX90+CS7 ist mit einer RS232 Verbindung ausgestattet. Ein AdeptV+ Programm empfängt gewünschte Gelenkpositionen über die RS232 Schnittstelle und sendet die aktuellen Gelenkpositionen des Roboters über diese Schnittstelle an den angeschlossenen PC.

Für Linux existiert eine Steuerungsoftware für den RX90. Diese implementiert das vom AdeptV+ Progrmm verwendete RS232 Protokoll und beinhaltet eine (Invers)Kinematik. Die Oberfläche ist in QT realisiert. Über die GUI kann der Benutzer manuell Positionen im kartesischen Raum oder in Gelenkwinkeln anfahren sowie Verfahrgeschwindigkeiten ändern oder Gelenkpositionen sowie kartesische Koordinaten auslesen. Diese QT-basierte Software soll im Rahmen dieser Arbeit verwendet werden und muss um eine Anbindung für OpenIGTLink erweitert werden, die die Möglichkeit bietet Positionen im kartesischen Raum anzufahren und auszulesen.

4.7. Kuka LWR IV

Der in Abbildung 21 gezeigte Roboter Kuka LWR IV nimmt eine Sonderstellung im Bereich der Knickarmroboter ein.

Abbildung 21: Kuka LWR IV

Der Kuka LWR[41] ist eine Entwicklung des DLR [42] und unterscheidet sich in vielerlei Hinsicht von anderen Knickarmrobotern. Der LWR IV wurde entwickelt, um mittels eines Roboters die menschliche Hand nachzuempfinden. So verfügt er mit sieben Freiheitsgraden über ein redundantes Gelenk und ist so in der Lage, ein und dieselbe Position und Orientierung unter verschiedenen Gelenkwinkeln anzufahren. Dies bietet vor allem Vorteile bei Arbeitsräumen, die wenig Platz für Roboterbewegungen bieten, oder in denen Menschen oder andere Maschinen arbeiten, die die Bewegungsfreiheit einschränken.

Ein weiterer Vorteil ist die Größe und das Gewicht des Roboters. Er ist vollkommen aus Kohlefaser gefertigt und hat ein Gewicht von 15 kg. Die Maximale Traglast bei niedrigen Verfahrgeschwindigkeiten beträgt 14 kg bei 0.05 mm Wiederholungsgenauigkeit. So ergibt sich ein Robotergewicht/Nutzlastverhältnis von nahezu 1:1. Die Nominallast beträgt 7 kg. Auch der LWR hat einen nahezu kugelförmigen Arbeitsraum. Der Radius beträgt hier circa 1150 mm.

Schließlich unterscheidet sich der LWR auch in elektronischer Hinsicht deutlich von anderen Industrierobotern. Konventionelle Industrieroboter wie der RX90 können ihre Gelenkpositionen z.B. über Impulsdrehgeber lesen. Der LWR ist zusätzlich mit Kraftmomentensensorik in jedem Gelenk ausgestattet. Über diese Kraftmomentensensorik können die Kräfte und Momente in jedem Gelenk ausgelesen werden. Der Roboter kann deshalb auf äußere Kräfte reagieren und damit in allen Gelenken nachgeben (in Anlehnung an die menschliche Hand), oder komplett frei beweglich (durch externe Kräfte) sein. Die Flexibilität für jedes Gelenk oder auch für die Achsen in kartesischen Koordinaten kann vom Benutzer definiert werden. Die Sensorik sowie die Motorik arbeiten mit einer Frequenz von 3 kHz. Wird nun die Steifigkeit einzelner Gelenke oder des gesamten Roboters auf null gesetzt, so kann der Roboter durch äußere Krafteinwirkung an eine beliebige Position in seinem Arbeitsraum beweget werden. Der Roboter misst hierzu die Kräfte und Momente in den Gelenken und führt die Antriebe entsprechend nach. Aufgrund der enormen Geschwindigkeit von Sensorik und Motorik sowie des Robotergewichts und des redundanten Gelenkes ergeben sich vielfältige Anwendungsmöglichkeiten für den LWR IV. So kann in Anwendungsbereichen, in denen Mensch und Maschine im gleichen Arbeitsraum arbeiten, die Verletzungsgefahr durch permanente Kraftmessungen deutlich reduziert werden. Der LWR eignet sich aber auch für humanoide Roboter wie Rollin‘ Justin[43], bei dem zwei LWR als Arme zum Einsatz kommen.

Für diese Arbeit bietet sich der LWR vor allem aufgrund der internen Kraftmomentensensorik an. So kann der Roboter zum Einlernen der Positionen in einen Modus gebracht werden, in dem alle Gelenke flexibel sind. In diesem Modus kann der Roboter durch mechanische Kräfte von außen geführt werden (Kräfte werden über Sensorik gemessen und Motorik nachgeführt) und an gewünschten Positionen einfach die Position und Orientierung im kartesischen Koordinatensystem gelesen werden. Der Roboter unterstützt außerdem einen Modus, bei dem auf einer beliebigen flexibel konfigurierten Achse ein benutzerdefinierter Anpressdruck eingestellt werden kann. Unter Verwendung dieses Modus kann der Roboter einer Oberfläche folgen. Dies lässt sich in der Ultraschalltomographie nutzen, um die Sonde flexibel über Weichgewebe zu führen und den Druck konstant zu halten ohne den Patienten zu gefährden.

---

[41] http://www.dlr.de/rm/desktopdefault.aspx/tabid-3803/ (letzter Zugriff: 01.12.2010) und Kuka LWR IV Anleitung

[42] http://www.dlr.de/ (letzter Zugriff: 01.12.2010)

[43] http://www.dlr.de/rm/desktopdefault.aspx/tabid-5471/8991_read-16694/ (letzter Zugriff: 01.12.2010)

Auch der Kuka LWR ist mit einer in Abbildung 22 zu sehenden externen Steuerung ausgestattet. Diese Steuerung ist im Vergleich zur CS7 des RX90 deutlich kompakter und benötigt aufgrund der geringen elektrischen Leistung des LWR ausschließlich eine konventionelle 230 Volt AC Stromversorgung. Kuka verwendet für alle Industrieroboter ein einheitliches Steuerungssystem, das hinsichtlich Leistungselektronik und Ausstattungsumfang an den verwendeten Roboter angepasst wird. Kuka Steuerungen beinhalten einen Rechner, der mit Windows XP embedded arbeitet und über das in Abbildung 23 zu sehende Kuka Control Panel gesteuert wird.

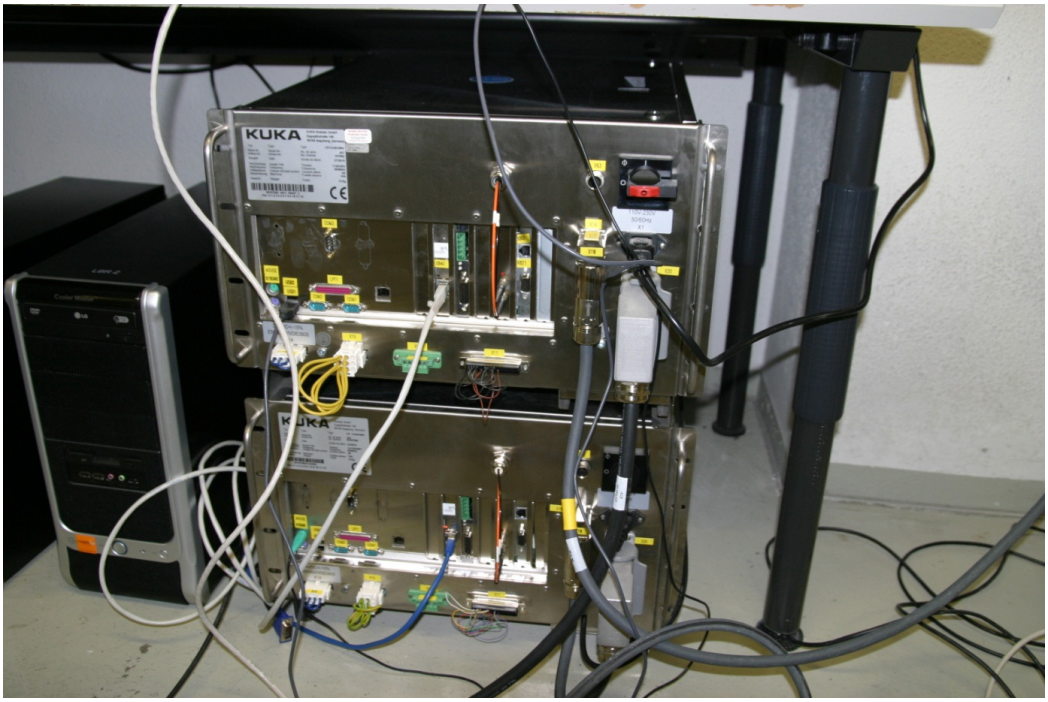

Abbildung 22: Steuerungen von 2 Kuka LWR IV

Abbildung 23: Kuka Control Panel

Über das Kuka Control Panel können Programme geladen werden oder der Roboter manuell gesteuert werden.

Eine Steuerung über externe Geräte wurde von Kuka mittels des FRI (Fast Research Interface) verwirklicht. Dieses Protokoll dient der Hochgeschwindigkeitsübertragung von Kommandos zum und vom Roboter und kann über die Ethernet Netzwerkschnittstelle arbeiten. Mittels des FRI können alle Funktionen des LWR kontrolliert werden. Im Rahmen eines anderen Projekts am Institut wurde eine CORBA Schnittstelle für das FRI entwickelt. MEX Files (Matlab kompilierte Bibliotheken) ermöglichen den Zugriff auf diese CORBA Schnittstelle und somit den simultanen Zugriff auf das FRI und damit den LWR aus mehreren Matlab Instanzen, die auf verschiedenen Rechnern laufen können.

5. Realisierung

Kapitel 5 beschäftigt sich mit der Realisierung des Gesamtsystems. Eingangs wird Fokus auf die Verknüpfung der Systemkomponenten und das modulare Gesamtkonzept gelegt. Die einzelnen Softwarebestandteile des Ultraschalltomographen werden vorgestellt. Anschließend werden die für Slicer und Matlab programmierten Kernkomponenten beschrieben und auf die benötigten Algorithmen eingegangen.

5.1. Verknüpfung der Systemkomponenten

Um eine optimale Flexibilität bei der Konfiguration und Benutzung des Ultraschalltomographen zu ermöglichen, wurde ein modulares Konzept gewählt. Deshalb wurden wie in Kapitel 4 bereits beschrieben alle Komponenten über Netzwerkprotokolle wie OpenIGTLink, oder CORBA miteinander verbunden. Dies bietet Vorteile beim Austausch einzelner Systemkomponenten und bei der Konfiguration des Systems. Es besteht die Möglichkeit, beliebige Ultraschallgeräte oder Roboter zu verwenden, sofern diese mit den hier verwendeten Protokollen ausgestattet werden. Besondere Vorteile ergeben sich in Zusammenhang mit der Steuerung der Geräte und der Verarbeitung der Parameterdatensätze. Da durchweg OpenIGTLink verwendet wurde, ist es möglich, die Robotersteuerung anstatt in Matlab auch in einem Modul für Slicer oder einer Stand-Alone Lösung zu verwirklichen. Genauso bietet sich die Möglichkeit, die Nachverarbeitung nicht mit Slicer, sondern mit Matlab vorzunehmen. Zu jedem Zeitpunkt des Projektes war es so möglich, auf geänderte Anforderungen oder Rahmenbedingungen zu reagieren.

Als Entwicklungsumgebungen kamen für Linux-basierte Systemkomponenten Eclipse Ganymed mit GCC 4.4 und Matlab zum Einsatz. Des Weiteren wurden eine modifizierte Version von OpenIGTLink (siehe 5.1.1), QT und die Slicer zugrundeliegenden Bibliotheken verwendet (MRML, ITK, VTK, etc.). Programmiert wurde in TCL/TK, Matlab, C++ und C.

Als Entwicklungsumgebungen für Windows-basierte Systemkomponenten kamen Visual Studio 2008 und Matlab zum Einsatz. Verwendete Bibliotheken waren eine modifizierte Version von OpenIGTLink (siehe 5.1.1), und das DiPhAS API. Programmiert wurde in Matlab, C, C++, C# und C++/CLI[44].

[44] http://msdn.microsoft.com/en-us/magazine/cc163681.aspx (letzter Zugriff: 01.12.2010)

5.1.1. Modifikation von OpenIGTLink

OpenIGTLink bietet wie bereits in Kapitel 4 erwähnt Möglichkeiten zur Steuerung von bildgebenden Modalitäten und Robotern. Da allerdings erwünscht war, dass Konfigurationsparameter für das DiPhAS systemübergreifend ausgetauscht werden sollen, war eine Erweiterung des OpenIGTLink Protokolls nötig. Zu diesem Zwecke wurde die OpenIGTLink Nachricht DiphasControlMessage eingeführt.

Der Body der Nachricht enthält folgende Felder:

Feldkürzel	Feldname	Dateityp	Beschreibung
O	operationMode	int	Arbeitsmodus für DiPhAS (8 bit, 16 bit, single Line Data)
D	depth_m	float	Eindringtiefe in Metern
E	Excitation	int	Code für vordefinierte Pulsform (siehe DiPhAS API)
EH	excitationHighLevelPercentage	float	Maximalwert für selbstdefinierte Pulsformen in Prozent (0.0-1.0)
EL	excitationLowLevelPercentage	float	Minimalwert für selbstdefinierte Pulsformen in Prozent (0.0-1.0)
F1-4	foc_m	float[4]	Fokussierungen für DiPhAS in Metern als Array
G	Gain	byte	Verstärkungsfaktor in Byte
T1-8	Tgc	byte[8]	eindrigtiefen-abhängiger Verstärkungsfaktoren in Byte als Array
V	Voltage	int	Erregunsgsspannung für Ultraschallsonde
S	scanControl	int	Zum Starten (1) oder Stoppen(0) der Bildakquisition

Tabelle 2: Aufbau der DiphasControlMessage Nachricht

Mittels der DiphasControlMessage Nachricht kann ein kompletter Datensatz, der zur Konfiguration des DiPhAS im B-Mode notwendig ist, über OpenIGTLink ausgetauscht werden. Außerdem kann die Bildakquisition über S (scanControl) gestartet und gestoppt werden.

Um eine korrekte Makefile erstellen zu können, mussten die neu generierten Quellcodedateien der Konfigurationsdatei für CMake hinzugefügt und CMake erneut ausgeführt werden.

Anschließend konnte die modifizierte Version von OpenIGTLink mit Visual Studio, oder GCC kompiliert werden

5.1.2. Anbindung des RX90 an OpenIGTLink

Die Steuerungssoftware des RX90 wurde wie erwähnt am Institut bereits in der Vergangenheit implementiert. Um den RX90 mittels OpenIGTLink ins System einzubinden, wurden neue Klassen erstellt, die über Eventsystem von QT in die Anwendung eingebunden wurden. Der RX90 sendet über sein serielles Protokoll die aktuellen Gelenkwinkel und Statusmeldungen, wenn eine Zielposition erreicht ist. Außerdem kann er über dieses Protokoll gewünschte Verfahrgeschwindigkeiten und Zielpositionen empfangen. Wie bereits in Kapitel 4 erwähnt beinhaltet die Steuerungssoftware eine Invers(Kinematik), die Gelenkwinkel bzw. kartesische Koordinaten berechnen kann. Um vollständige Kontrolle über den RX90 via OpenIGTLink zu erlangen, müssen von der Steuerungssoftware homogene Koordinaten und Statusmeldungen über OpenIGTLink und das RX90 Protokoll empfangen und gesendet werden können.

Um die Steuerung möglichst flexibel zu halten, wurde die Anbindung an OpenIGTLink über vier Ports verwirklicht. Die Steuerung fungiert hierbei als Server. Jeder einzelne Port ist hierbei als Einwegverbindung konzipiert. Durch die Auftrennung in vier Ports für Status empfangen, Status senden, Position empfangen und Position senden können unterschiedliche Clients mit dem RX90 verbunden werden. So kann z.B. via Matlab eine Position an den RX90 gesendet werden, der dann wiederrum nach Erreichen der Zielposition seinen „erreicht“-Status an das DiPhAS senden kann. Während dieser Arbeit wurden alle vier Ports ausschließlich mit Matlab verbunden.

Die Steuerung wurde so realisiert, dass bei Empfang einer Nachricht über das RX90 Protokoll ein QT Event ausgelöst wird. Mittels dieser Events wird die Anzeige der Positionen auf der GUI realisiert. Außerdem kann über das ausgelöste Event die Weiterleitung der Positionen und Statusmeldungen an OpenIGTLink erfolgen. Bei Empfang einer „erreicht“-Nachricht vom Roboter wird der Versand einer OpenIGTLink Status Nachricht mit Status-Code 5 eingeleitet, der im Falle der Robotersteuerung für Position erreicht steht. Ebenso wird bei Empfang der Gelenkwerte des Roboters der Versand der homogenen Koordinaten, die durch die Kinematik berechnet wurden, über OpenIGTLink (TransformMessage) initiiert.

Die einzelnen Klassen, die für die OpenIGTLink Verbindungen zuständig sind, sind als eigenständige QThreads ausgeführt und arbeiten deshalb unabhängig vom Hauptprogramm. OpenIGTLink Verbindungen, die sich um den Empfang von Nachrichten kümmern, verfallen nach Aufbau der Verbindung in einen Wartezustand. Sobald eine Nachricht empfangen wurde erwachen diese aus dem Wartezustand, überprüfen die empfangene Nachricht und lösen, falls diese korrekt ist, Events im Hauptprogramm aus. Wird eine TransformMessage empfangen, so wird die Berechnung der Gelenkwerte und der Versand der Gelenkwerte über die RS232 Schnittstelle an den Roboter initiiert. Wird eine StatusMessage mit Status Code 7 (setzen einer neuen Robotergeschwindigkeit) empfangen, so wird der Sub-Code der Status Nachricht ausgelesen. Dieser Sub-Code muss im Bereich zwischen 0 und 100 liegen und gibt den Prozentsatz der maximal erreichbaren Robotergeschwindigkeit an, mit der der Roboter fahren soll. Dieser Sub-Code wird über das Eventsystem von QT an die RS232 Schnittstelle weitergeleitet.

Abbildung 24 zeigt die Kontrollanwendung für den RX90 mit verbundenem Roboter und verbundenen OpenIGTLink Ports.

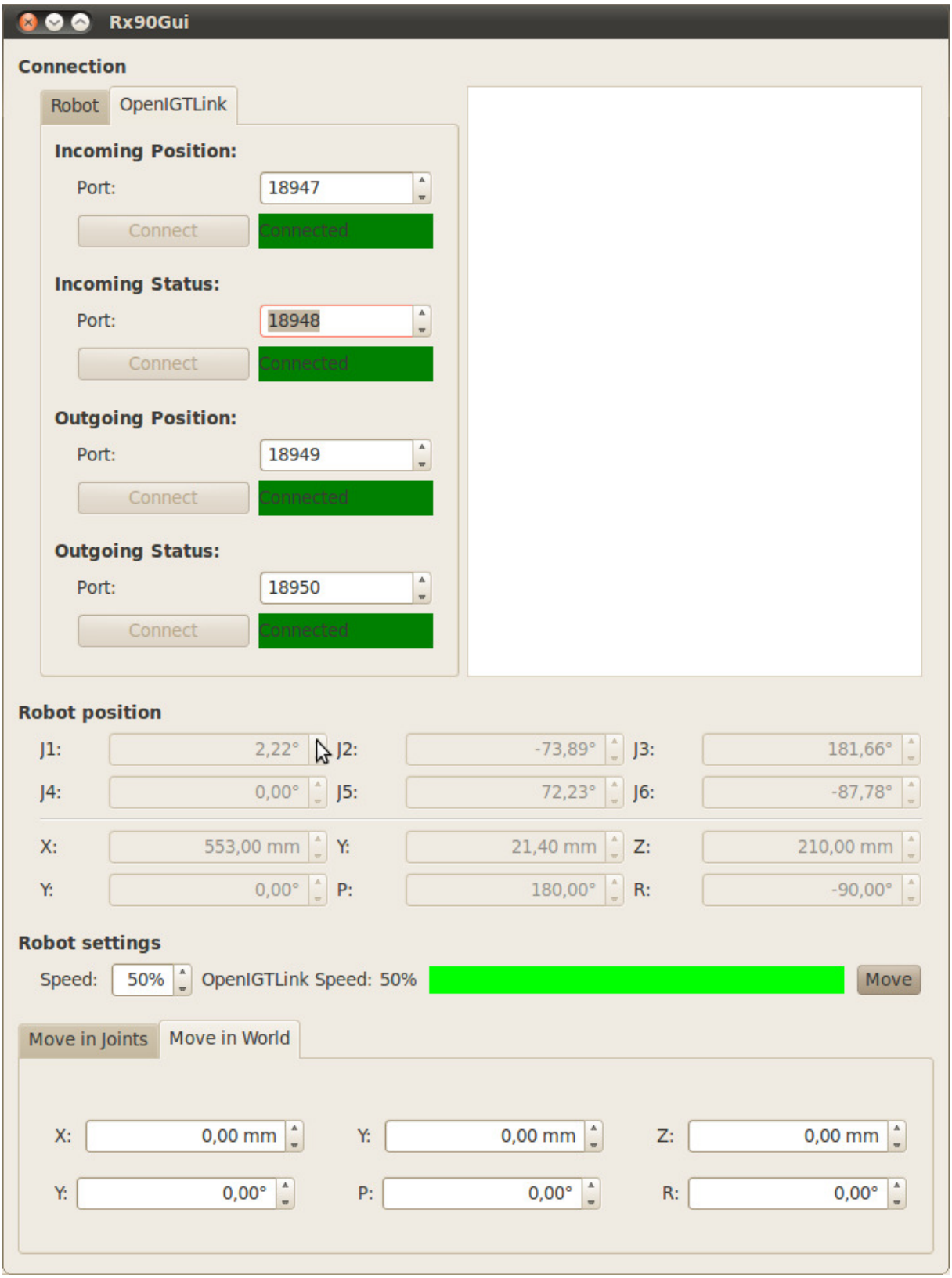

Abbildung 24: RX90Control mit verbundenen OpenIGTLink Ports

5.1.3. Anbindung von Matlab an OpenIGTLink

Für Matlab existiert wie bereits erwähnt eine Implementierung von OpenIGTLink. Diese Implementierung befindet sich noch in einem sehr frühen Stadium und eignet sich deshalb nur mit Modifikationen für diese Arbeit. Die offizielle Implementierung arbeitet ohne Threads und unterstützt die Nachrichtentypen TRANSFORM und IMAGE. Für den Ultraschalltomographen ist eine Unterstützung der STATUS Message und Threads von essentieller Bedeutung. Matlab soll später zur Kontrolle der Modalitäten Status Nachrichten an das DiPhAS senden und von Slicer empfangen können.

OpenIGTLink arbeitet mit Sockets und überschreibt bereits empfangene Nachrichten nicht. Empfangene Nachrichten werden gepuffert. Wie häufig die Robotersteuerung Nachrichten sendet, hängt von vielen Variablen ab und kann nicht eindeutig bestimmt werden. Würde mit einer Implementierung ohne Threads gearbeitet werden, könnte nicht gewährleistet werden, dass die aktuelle Roboterposition tatsächlich als nächstes ausgelesen wird. Die Nachrichten werden nacheinander aus dem Puffer gelesen. Wenn der Roboter während des Umpositionierens z.B. 200 Positionen über OpenIGTLink sendet, würde Matlab nach dem Lesen der ersten Position aus dem Puffer einen veralteten Wert empfangen, während der Roboter bereits mehrere neue Positionen eingenommen hat. Ähnliche Probleme ergeben sich bei Empfang von Statusnachrichten. Der Roboter sendet nicht ausschließlich die „erreicht“-Nachricht über das RS232 Protokoll an die Steuerung, sondern außerdem mehrfach Nachrichten, die indizieren, dass der Roboter die Hälfte der Strecke gefahren ist oder dass eine Position nicht erreichbar ist. Auch hier können veraltete Werte ausgelesen werden. Außerdem ergibt sich zusätzlich das Problem, dass Matlab ebenfalls in einen Wartezustand verfällt, wenn keine Werte mehr auf dem Socket verfügbar sind. Matlab kann erst wieder bedient werden und reagieren, wenn die Nachricht empfangen wurde, auf die gewartet wurde.

Bei Verwendung eines Threads kann dieser immer auf neue Nachrichten warten und bei Empfang einer neuen Nachricht die aktuellen Informationen aus der Nachricht für Matlab vorhalten. So kann gewährleistet werden, dass beim Lesen der Nachricht aus Matlab heraus immer die aktuellste Nachricht verwendet wird. Ist keine neue Nachricht verfügbar, verfällt Matlab nicht in den Wartezustand, der Thread hingegen schon. Matlab kann aber weiterhin benutzt werden.

Im Rahmen dieser Arbeit wurden mex-Files entwickelt, die dieses Verhalten für TRANSFORM und STATUS Message implementieren. Die Original mex-File, die ohne Threads arbeitet, wurde außerdem um Unterstützung für STATUS und POSITION Message erweitert.

Bei der Implementierung ergaben sich einige Schwierigkeiten. Für Linux wurden POSIX Threads in der pThread Implementierung verwendet. Diese Implementierung existiert allerdings nicht in identischer Form für Windows. So musste die Unterstützung für Threads unter Windows auf Basis des Windows API realisiert werden.

Aufgrund der nicht plattformübergreifenden MEX-Files für die Anbindung des Kuka LWR IV über CORBA musste die Ausführung der Mex-Files und damit Matlab unter Windows erfolgen.

5.1.4. DiPhAS Networker Client und C++/CLI Wrapper

Auch das DiPhAS musste via OpenIGTLink in das Setup eingebunden werden. Deshalb wurde mittels des DiPhAS API, die auf .NET basiert, eine Anwendung in C# programmiert, die die akquirierten Bilder des DiPhAS in Echtzeit darstellt, sich über OpenIGTLink mit anderen Anwendungsteilen verbinden kann , eine Rekonfiguration über OpenIGTLink im laufenden Betrieb zuıässt und auf Anfrage das aktuelle Bild über OpenIGTLink versendet.

Abbildung 25 zeigt diese Anwendung in der finalen Konfiguration.

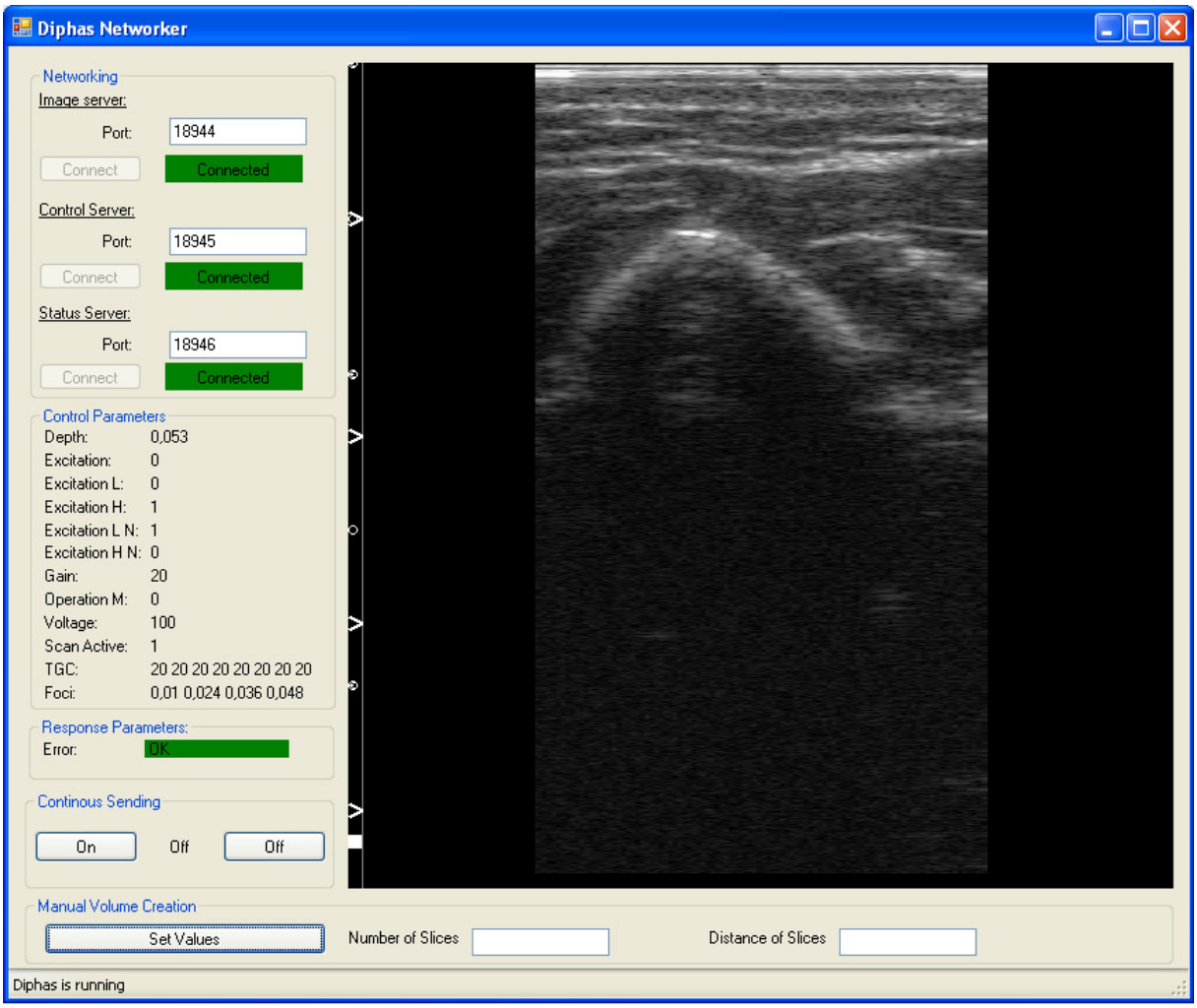

Abbildung 25: DiPhAS Networker

DiPhAS Networker ist eine Windows Forms Anwendung, die in C# entwickelt ist. Die Oberfläche ist komplett mit dem grafischen Designer von Visual Studio 2008 entwickelt. Die GUI ist komplett von der Anwendungslogik getrennt. Bei Erstellung der GUI Klasse werden die einzelnen Methoden zur Manipulation der Oberfläche an die Logik übergeben und können anschließend in dieser aufgerufen werden. Dies geschieht in .NET Programmiersprachen mittels Delegaten. Delegate sind vergleichbar mit den in C und C++ verwendeten Funktionenzeigern. In der Logik muss die an diese übergebene Methode (aus der GUI) einem Delegatobjekt zugeordnet werden. Über dieses Delegatobjekt kann die Methode in der GUI aus der Logik heraus aufgerufen werden. Hierzu wird das Delegatobjekt auf

dieselbe Art und Weise aufgerufen, wie die Methode in der GUI aufgerufen werden würde. Diesem Objekt werden die Parameter für die aufzurufende Methode mitgegeben. Für einfache Programmstrukturen kann so eine einfache Trennung zwischen Oberfläche und Logik aufgebaut werden ohne viel Overhead zu produzieren.

Das DiPhAS API stellt vielfältige Klassen zur Kontrolle der DiPhAS Hardware und zur Nachbearbeitung der Signale zur Verfügung. Es stehen Imaging Klassen zur Verfügung, die Konvertierungen von DiPhAS Datensätzen in Standardformate ermöglichen oder die Darstellung der akquirierten Bilder im B-, M-, oder Dopplermodus beeinflussen. So können z.B. Labels für Eindringtiefe und Fokus ein- oder ausgeblendet werden.

Außerdem sind Klassen für den Import und Export von Daten oder Postprocessing vorhanden. So können z.B. Bildserien als AVI-Videos exportiert werden, DICOM Datensätze geladen oder versendet werden. Auch die Integration der Ultraschallbilder in DICOM Datensätze ist möglich. Die Postprocessing Möglichkeiten sind ebenfalls umfangreich. So können mittels des API Filter implementiert werden, die direkt auf Rohdaten, Hochfrequenzdaten oder Bildern arbeiten können. Nativer Support für Trackingsysteme und Volumenrekonstruktion mittels Bildserien ist vorhanden.

Für die Kontrolle der Hardware stehen Klassen für den High Level Zugriff zur Verfügung, mit denen Sende- und Empfangsparameter konfiguriert werden können. Außerdem existieren Klassen mit denen die Hardware auf tieferer Ebene gesteuert werden kann. Hiermit kann der Benutzer deutlich stärker in die Signalgenerierung und -akquisition eingreifen.

Bei DiPhAS handelt es sich um ein sehr neues System. Das API ist zu weiten Teilen spezifiziert, aber noch nicht komplett implementiert. Die für diese Arbeit notwendigen Klassen zum Highlevel Zugriff sind vollständig vorhanden. Leider verursachte das Ausblenden der Labels für Eindringtiefe und Fokus einen Fehler im DiPhAS API, der aus unbekannten Gründen vom IBMT bisher nicht behoben werden konnte. Dies erforderte einen Workaround, der in Kapitel 5.4.1. näher beschrieben ist.

Zum Zugriff auf die DiPhAS Hardware muss ein DiPhASControl Objekt angelegt werden. Über dieses Objekt kann das DiPhAS konfiguriert werden. Es können Statusmeldungen von der DiPhAS Hardware empfangen, Parameter an diese übermittelt oder aktive Sonden gewählt werden. Um Eindringtiefe, Fokussierungen, Verstärkungsfaktoren etc. zu konfigurieren, werden die gewünschten Werte in einem ControlDeviceParameters Objekt abgelegt und dieses an das DiPhASControl Objekt übergeben. Die DiPhAS Hardware wird nach Übergabe dieses Objektes mit sofortiger Wirkung neu konfiguriert.

Die Aufnahmegeschwindigkeit des DiPhAS richtet sich nach der aktuellen Konfiguration. Je nach gewählter Konfiguration können Geschwindigkeiten im Bereich weniger Frames bis hin zu 30 Frames pro Sekunde erreicht werden. DiPhASNetworker arbeitet ausschließlich im B-Modus, da diese Arbeit sich nicht mit der Verarbeitung von Roh oder HF-Daten beschäftigt. Sobald ein neues B-Modus Bild vorliegt, wird dieses als Bitmap an ein in dem API definiertes Delegatobjekt übermittelt und anschließend in der diesem Objekt zugeordneten, benutzerdefinierten Methode verarbeitet.

Die Anbindung an OpenIGTLink gestaltete sich im Falle des DiPhAS Networker komplizierter als bei C++ Anwendungen. Für die .NET Plattform existiert leider bisher keine OpenIGTLink Implementierung. .NET Sprachen arbeiten alle auf Basis der Common Language Runtime (CLR). Die

einzelnen Sprachen erzeugen lediglich Zwischencode in der Common Intermediate Language (CIL), der anschließend von der CLR in den finalen Anwendungscode übersetzt wird.

Microsoft bietet mit C++/CLI eine Version von C++ an, die sich zum Teil erheblich von üblichem C++ Code unterscheidet. C++/CLI hält sich an den Common Language Infrastructure-Standard (CLI). Deshalb kann aus C++/CLI Code ebenfalls gültiger CIL Code erzeugt werden. In C++/CLI Projekte können normale C++ Bibliotheken importiert werden. C++/CLI eignet sich somit hervorragend, um Wrapper zu schreiben, die die Verwendung von Bibliotheken, die nicht für .NET entwickelt wurden, dennoch in .NET ermöglichen.

In C++/CLI wird strikt zwischen Managed und Unmanaged Classes unterschieden. Managed Classes sind hierbei Klassen, die später in der CLR laufen sollen. Unmanaged Classes hingegen arbeiten mit Code, der direkt auf der CPU läuft. Klassen die in der CLR laufen, müssen deshalb explizit als solche gekennzeichnet werden. Methodenköpfe in Managed Classes dürfen keine C++ Pointer, sondern nur .NET kompatible Handles enthalten, die anders wie Pointer mit ^ markiert werden. In Managed Classes darf dennoch auf Unmanaged Methoden aus Unmanaged Classes zugegriffen werden. C++/CLI verwendet hierzu P/Invoke[45], das sich um den Aufruf des nativen C++ Codes und das Wrapping der Datentypen kümmert.

Auch bei DiPhAS Networker wurde die Verbindung zu anderen Anwendungsteilen mittels mehrerer OpenIGTLink Verbindungen realisiert. Bei DiPhAS Networker wurden drei Server verwendet. Die Verbindungen behandeln den Empfang von Konfigurationen für die DiPhAS Hardware, den Empfang von Statusmeldungen und den Versand der Bilder. Bei dem während dieser Arbeit entstandenen Wrapper wurden für jeden Nachrichtentyp einzelne Wrapperklassen programmiert. Jeder Server empfängt oder sendet jeweils ausschließlich einen der Nachrichtentypen. Für jeden Nachrichtentyp existieren eine Managed Class und eine Unmanaged Class, wobei Letztere von der Klasse unmanagedServer erbt. Soll nun einen Server geöffnet werden, so muss die Wrapper-Klasse verwendet werden, die den gewünschten Nachrichtentyp wrappt, und über diese der Server gestartet werden. Der Wrapper ist als dynamisch verlinkte Bibliothek (dll) realisiert und kann in .NET Anwendungen importiert werden. Über die Methoden in der Managed Class für den entsprechenden Nachrichtentyp kann der Server für OpenIGTLink konfiguriert, gestartet und verbunden werden. Außerdem können Nachrichten erstellt und über den Server versendet werden. Obwohl der Aufruf von Unmanaged Methoden aus Managed Code heraus möglich ist, muss beachtet werden, dass die Datentypen zwischen CLR und nativem C++ Code nicht kompatibel sind. Microsoft nennt den Vorgang des Wrappens Marshaling. Marshaling von einfachen Dateitypen wie Integer wird von C++/CLI automatisch mittels P/Invoke vorgenommen. Insbesondere Pointer müssen mittels der Klasse System::Runtime::InteropServices::Marshal aber von Hand konvertiert werden.

Während der Aufruf von Unmanaged Code aus .NET verhältnissmäßig einfach zu realisieren ist, ergeben sich beim Empfang von Nachrichten über den Server Probleme. Um dem Hauptprogramm zu signalisieren, dass neue OpenIGTLink Nachrichten vorliegen, muss Managed Code aus Unmanaged Code heraus aufgerufen werden. Zur Lösung dieses Problems können Delegaten verwendet werden. C++/CLI kann Delegaten und andere Funktionen der .NET Umgebung wie z.B. Garbage Collection nutzen. Um DiPhAS Networker das Vorliegen neuer Nachrichten zu signalisieren, wird von diesem die

[45] http://www.microsoft.com/germany/msdn/library/net/SoArbeitetNETMitWin32UndCOMZusammen.mspx?mfr=true (letzter Zugriff: 01.12.2010)

„listen“-Methode der Managed Class der entsprechenden Nachricht aufgerufen. Aus der Managed „listen“-Methode wird die „listen“-Methode des Unmanaged Teils aufgerufen und gewartet, bis eine Nachricht empfangen wurde. Diese wird anschließend entschlüsselt und die relevanten Parameter an den Managed Teil des Wrappers übergeben, wo der Datensatz über den Delegaten an das Hauptprogramm übermittelt wird. Um den DiPhAS Networker nicht ebenfalls in den Wartezustand des Servers zu versetzen, werden vom Hauptprogramm zur Steuerung der Server Threads verwendet. So wartet beim Aufruf der Listen Methode nicht der komplette DiPhAS Networker, sondern ausschließlich der aufrufende Thread.

Nach dem Start von DiPhAS Networker und dem Aufbau der Verbindungen zu anderen Modalitäten erfolgt die Konfiguration durch externe Komponenten (z.B. Matlab, oder Slicer). DiPhAS Networker muss hierzu eine DiphasControlMessage empfangen, die einen gültigen Datensatz enthält. Der Wrapper übermittelt diesen Datensatz an das Hauptprogramm. Dort wird überprüft ob alle Parameter innerhalb des erlaubten Wertebereichs liegen. Ist dies der Fall werden die Parameter an die DiPhAS Hardware übermittelt, die danach kontinuierlich Bilder liefert. Diese Bilder haben eine vom Benutzer definierbare Größe. Während dieser Arbeit wurde eine Auflösung von 600x600 Pixeln verwendet. In Abbildung 25 ist ersichtlich, dass nicht der komplette Inhalt des Bildes Ultraschalldaten erhält. Die Breite und Höhe der Nutzinformation innerhalb des Bildes ist abhängig von der Eindringtiefe. Das DiPhAS API berechnet die Höhe und Breite des Bildes so, dass alle Pixel quadratische Abmessungen haben. Bei einer Strahlbreite von 29.1 mm müsste die Eindringtiefe also auf 29.1 mm gesetzt werden, um ein quadratisches Ultraschallbild zu erzeugen.

Empfängt der Server für Status Nachrichten eine Status Nachricht mit Code 1 (OK), werden die Anteile des aktuellen Bildes entfernt, die keine Nutzinformationen enthalten (siehe Kapitel 5.4.1.). Außerdem liegt das Graustufenbild als RGB Information vor. In Graustufenbildern haben alle Kanäle eines Pixels denselben Wert. Zur Konvertierung des RGB Bildes in das OpenIGTLink Format genügt es deshalb zwei beliebige der drei Kanäle jedes Pixels zu entfernen und anschließend alle Pixel nacheinander in ein Array zu schreiben. Dieses Array wird über den Server für den Versand von IMAGE-Nachrichten als OpenIGTLink IMAGE-Nachricht an 3D Slicer übertragen.

DiPhAS Networker kann außerdem auf Wunsch einen kontinuierlichen Bildstrom über OpenIGTLink liefern.

5.1.5. Ablauf des Kontrollflusses

Um einen 3D-Datensatz zu erstellen, müssen die Komponenten miteinander verbunden werden. Kern der Anwendung ist hierbei Matlab. Matlab übernimmt die Synchronisation und Steuerung der Geräte. In Abbildung 26 ist zu sehen, wie die Geräte interagieren. Alle Verbindungen mit Ausnahme der Anbindung des LWR IV an Matlab (CORBA) und der Verbindungen innerhalb Slicer (Eventsystem) sind mit OpenIGTLink realisiert. Die verwendeten Nachrichtentypen und Richtungen der Nachrichtenströme sind in der Abbildung markiert.

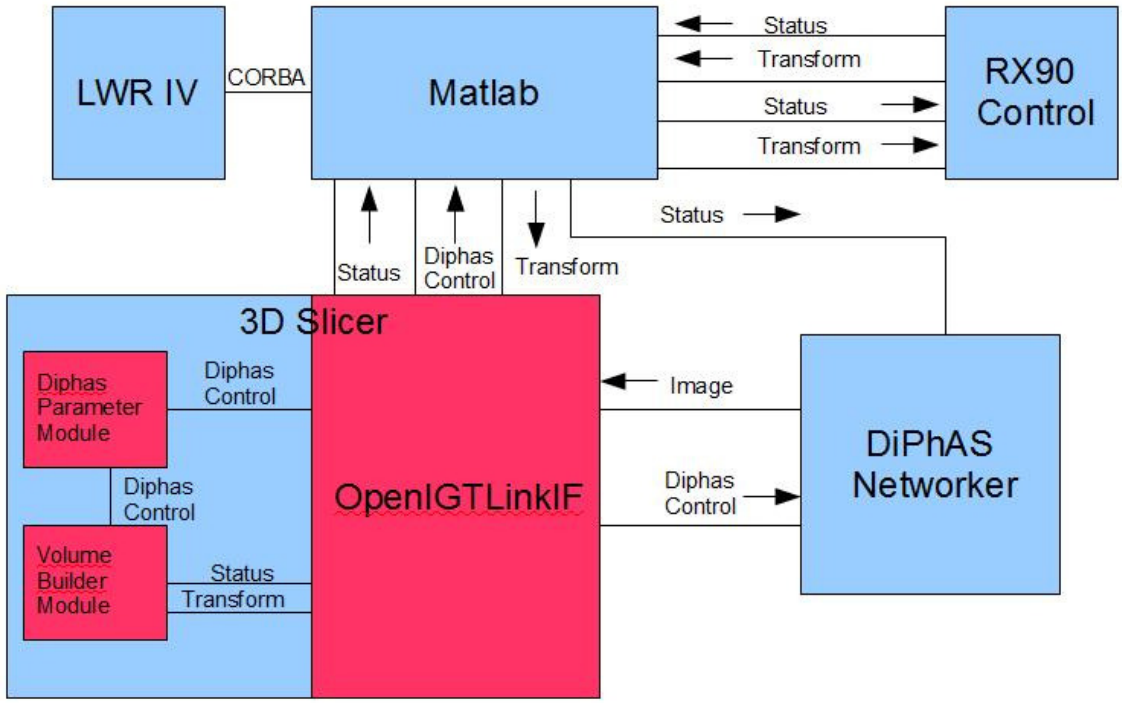

Abbildung 26: Interaktion der Systemkomponenten

Mittels des in Kapitel 5.3. beschrieben Matlab Skripts können lineare Trajektorien für den RX90 programmiert werden, oder Start und Zielpunkte für den LWR definiert werden, der seine Trajektorie zwischen den definierten Punkten während des Scans auf Grundlage der gemessenen Kräfte berechnet.

Zu Beginn eines Scans wird DiPhAS Networker über 3D Slicer (DiPhAS Parameter Module) konfiguriert. Über das Slicer Event System wird außerdem das Volume Builder Module über die DiPhAS Konfiguration in Kenntnis gesetzt. Das bei Änderungen im DiPhAS Parameter Module ausgelöste Event bewirkt den Versand des Parametersatzes an DiPhAS Networker und Matlab, welches die Konfiguration für die Berechnung von Rotationen benötigt.

Matlab berechnet zwischen Start und Zielpunkt Positionen in benutzerdefinierbaren Abständen. Eine berechnete Transformation wird über eine OpenIGTLink Status-Nachricht bzw. CORBA an den Roboter gesendet. Anschließend gibt Matlab den Robotern genug Zeit, um ihre neue Position anzufahren und wartet währenddessen. Ist die Position erreicht, sendet Matlab die aktuelle Robotertransformation an 3D Slicer und den Statuscode 1 (OK) an DiPhAS Networker.

DiPhAS Networker sendet nach dem Empfang einer Statusnachricht mit Statuscode 1 das aktuelle Bild, das an der neuen Roboterposition aufgenommen wurde, an 3D Slicer. Über das Slicer Eventsystem erreichen Transformation und Bild das Volume Builder Modul, wo sie miteinander verknüpft werden. Nach Empfang des Bildes initiiert Volume Builder Module über das das Slicer Event System und OpenIGTLinkIF den Versand einer OK Status Nachricht an Matlab.

Matlab wartet dabei nach dem Versand seiner Nachrichten an den Roboter und an DiPhAS so lange, bis die OK Nachricht von Slicer empfangen wurde und sendet anschließend die nächste berechnete Position an den Roboter. Die Vorgänge werden so lange wiederholt, bis der Zielpunkt erreicht ist. Am Ende eines Scans hält Volume Builder Module eine Liste von Bildern mit den zugehörigenen Transformationen und DiPhAS Konfigurationen und kann daraus einen 3D Datensatz berechnen.

5.2. 3D Slicer Module

Die Oberfläche für den Ultraschalltomographen sollte sehr einfach gehalten werden. Außerdem war eine nahtlose Integration in 3D Slicer erwünscht. Alle Operationen die sich mit der Bildverarbeitung und Erzeugung von Datensätzen beschäftigen, wurden in 3D Slicer realisiert. Alle Operationen, die Steuerung und Synchronisation der Modalitäten beeinflussen sowie die Berechnung der Roboterpositionen im kartesischen Raum übernehmen, werden in einem Matlab Skript ausgeführt.

Im Rahmen dieser Arbeit entstanden zwei 3D Slicer Module. DiPhAS Parameter Module dient ausschließlich zur Konfiguration des DiPhAS und kann unabhängig von Volume Builder Module benutzt werden. Volume Builder Module dient zur Nachbearbeitung der gesammelten 2D-Ultraschallbilder und zur Erzeugung der 3D Ultraschalldatensätze. Im Folgenden wird detailliert auf beide Module eingegangen.

5.2.1. DiPhAS Parameter Module

DiPhAS Parameter Module ist unabhängig von der Funktionalität des Ultraschalltomographen. DiPhAS Parameter Module erweitert 3D Slicer um ein Konfigurationsmodul für DiPhAS. Prinzipiell kann damit jedes Ultraschallgerät konfiguriert werden, sofern der Benutzer für dieses eine OpenIGTLink Anbindung entwickelt, die DiphasControlMessage Nachrichten empfangen kann. DiPhAS Parameter Module verwaltet ausschließlich Konfigurationen und akzeptiert keine Bilddatensätze.

Die Konfiguration des DiPhAS erfolgt über eine GUI, auf der Eindringtiefe, Anzahl und Position der Fokussierungen, Verstärkungsfaktor, Erregungsspannung, TGC, Operationsmodus, Pulsform sowie Minimal- und Maximalwerte für selbstdefinierte Pulsformen eingestellt werden.

Die Oberfläche des DiPhAS Parameter Module ist in Abbildung 27 abgebildet.

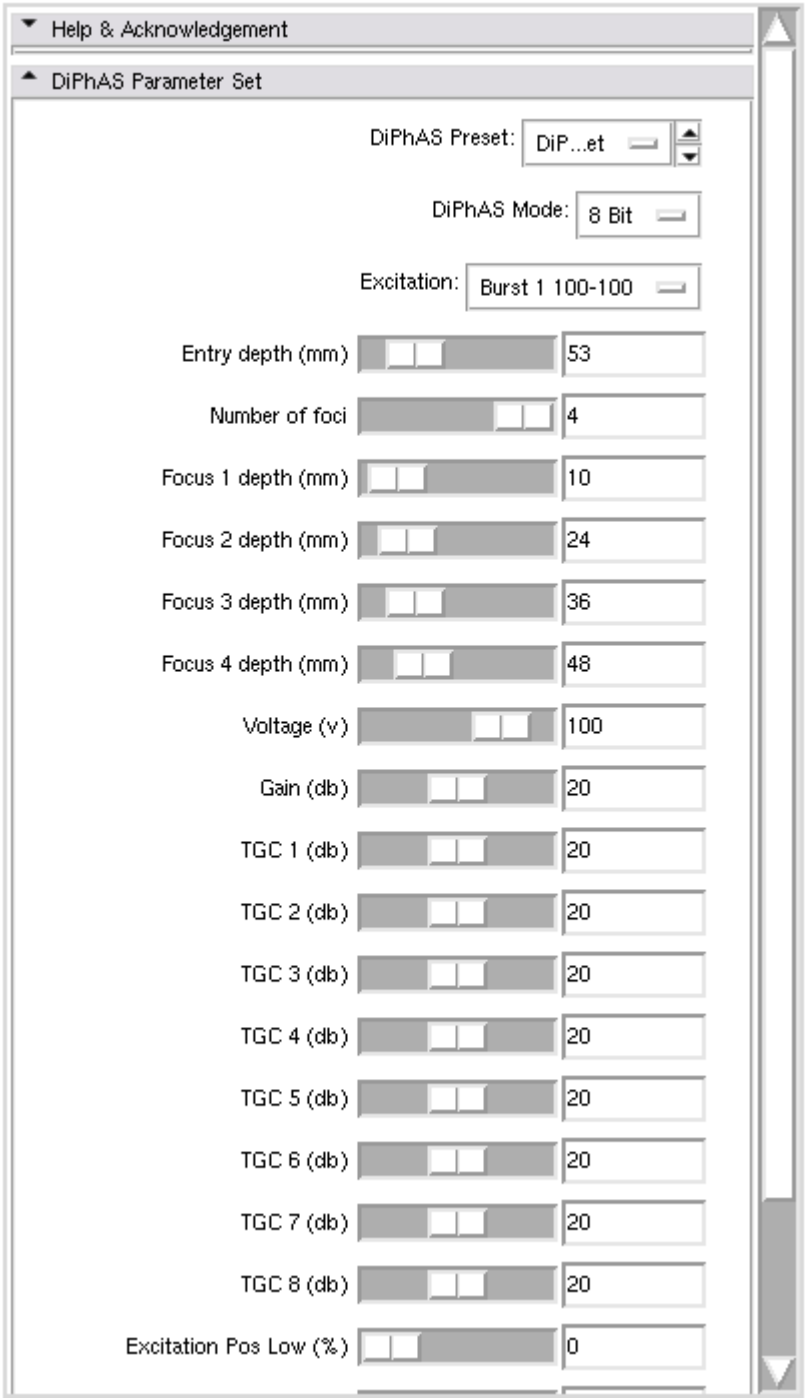

Abbildung 27: Oberfläche des DiPhAS Parameter Module

DiPhAS Parameter Module verwendet zum Versand der Nachrichten das Slicer Eventsystem. In 3D Slicer ist kein MRML Knoten definiert, der sich zur Aufnahme der Parameter für das Ultraschallgerät eignet. Deshalb wurde im Rahmen dieser Arbeit mit vtkMRMLDiphasParameterModuleNode ein Knoten definiert, der einen kompletten Parametersatz für DiPhAS aufnehmen kann. MRML Knoten die in Slicer verwendet werden sollen, müssen von der Klasse vtkMRMLNode erben und deren Methoden createNodeInstance, ReadXMLAttributes, WriteXMLAttributes, Copy und GetNodeTagName implementieren. CreateNodeInstance wird immer dann aufgerufen, wenn ein neuer Knoten des Typs benötigt wird. ReadXMLAttributes und WriteXMLAttributes definieren wie

der Knoten in einer Slicer-Szenen-Datei gespeichert oder daraus geladen wird. Copy liefert eine Kopie des Knotens. GetNodeTagName liefert den Knoten-Tag zurück. Dieser Knoten-Tag muss eindeutig sein. Slicer erkennt Module und Knoten über ihren Tag. Im Falle des vtkMRMLDiphasParameterModuleNode lautet der NodeTagName DiPhASParameters. Da es sich bei DiPhAS Parameter Module um ein Loadable Module handelt, kann direkt auf alle Teile des Slicer API zugegriffen werden. Um den neuen MRML Knoten in Slicer bekannt zu machen, genügt es deshalb einen neuen Knoten des entsprechenden Typs zu erzeugen und diesen anschließend bei der aktuellen MRML Szene mittels RegisterNodeClass zu registrieren.

```
vtkMRMLDiphasParameterModuleNode* diphasNode =
                        vtkMRMLDiphasParameterModuleNode::New();
this->GetMRMLScene()->RegisterNodeClass(diphasNode);
```

Der vtkMRMLDiphasParameterModuleNode ist fortan in jedem Modul von 3D Slicer verwendbar, sofern diesem die Quellen für vtkMRMLDiphasParameterModuleNode bekannt sind.

Ein ähnliches Problem ergibt sich bei der Konvertierung des vtkMRMLDiphasParameterModuleNode in eine OpenIGTLink Nachricht. Das Slicer Module OpenIGTLinkIF, das zur Kommunikation mit externen Komponenten verwendet wird, implementiert ausschließlich Nachrichtentypen aus dem OpenIGTLink Standard-Protokoll, in dem die DiphasControlMessage nicht enthalten ist. Glücklicherweise haben die Entwickler dieses Moduls ebenfalls ein Plugin System[46] entwickelt, das sich für die Lösung dieses Problems eignet. Um dieses zu nutzen, müssen die Quellen des OpenIGTLinkIF Moduls in das DiPhAS Parameter Module Projekt importiert werden. Anschließend kann während der Laufzeit von Slicer aus DiPhAS Parameter Module heraus über den Modulname von OpenIGTLinkIF die GUI Klasse von OpenIGTLinkIF angefordert werden.

```
vtkOpenIGTLinkIFGUI * igtlGUI =
        vtkOpenIGTLinkIFGUI::SafeDownCast(
        vtkSlicerApplication::SafeDownCast(
        this->GetApplication()) ->GetModuleGUIByName( "OpenIGTLink IF"));
```

Nach Aufruf von OpenIGTLinkIF (Enter-Methode) kann anschließend ein MessageConverter für den entsprechenden Knoten registriert werden, der benutzerdefinierte OpenIGTLink Nachrichten in MRML Knoten konvertieren kann. Auch die Konvertierung von MRML Knoten in OpenIGTLink Nachrichten ist möglich.

```
igtlGUI->Enter();
igtlGUI->GetLogic()->RegisterMessageConverter(diphasParameterConverter);
```

Damit dies möglich ist muss OpenIGTLinkIF eine Version der OpenIGTLink Bibliothek verwenden, die die gewünschte OpenIGTLink Nachricht implementiert. In unserem Falle muss die in 5.1.1. definierte DiphasControlMessage enthalten sein. Die Integration der um diese Nachricht erweiterten Bibliothek ist denkbar einfach. Hier machen sich vor allem die Vorteile von CMAKE bemerkbar. Es ist ausreichend die Quellen für die Nachricht zur OpenIGTLink Bibliothek hinzuzufügen, auf die OpenIGTLinkIF gelinkt ist, diese in der CMAKE Konfigurationsdatei einzutragen und CMAKE sowie Make für die OpenIGTLink-Bibliothek auszuführen. Anschließend müssen CMAKE und Make noch für

[46] http://www.na-mic.org/Wiki/index.php/OpenIGTLink/Slicer (letzter Zugriff: 01.12.2010)

OpenIGTLinkIF ausgeführt werden. Fortan kann OpenIGTLinkIF mit DiphasControlMessage verwendet werden.

Ein OpenIGTLinkIF Nachrichten-Konverter ist ebenfalls ein MRML Knoten. Alle Nachrichten-Konverter müssen von vtkIGTLToMRMLBase erben und die Methoden GetIGTLName, GetMRMLName, GetNodeEvents, CreateNewNode, IGTLToMRML und MRMLToIGTL implementieren.

GetIGTLName und GetMRMLName definieren jeweils die Strings zur Identifikation von MRML Knoten und OpenIGTLink Nachricht, die vom Konverter verwendet werden. GetNodeEvents definiert die Events, bei deren Auftreten der MRML Knoten in eine OpenIGTLink Nachricht konvertiert werden soll. CreateNewNode erzeugt einen neuen MRML Knoten. IGTLToMRML und MRMLToIGTL implementieren die Funktionalität des Konverters. MRMLToIGTL liest die Parameter aus MRML Knoten und ordnet diese den entsprechenden Feldern einer OpenIGTLink Nachricht zu, die anschließend gepackt und versendet wird. IGTLToMRML liest die Parameter in einer OpenIGTLink Nachricht und erzeugt einen MRML Knoten, der diese enthält. Wurde über die entsprechende OpenIGTLink Verbindung, die in OpenIGTLinkIF „Connector“ genannt wird, bereits ein Knoten erstellt, wird dieser nur aktualisiert. Das bedeutet, dass z.B. beim ersten Empfang eines Bildes ein MRML Knoten mit den Bildinformationen erstellt und bei jedem weiteren Empfang eines Bildes die Bildinformation im Knoten aktualisiert wird.

3D Slicer enthält keinen MRML Knoten für Statusnachrichten. Dieser wird allerdings für das in Kapitel 5.2.2. vorgestellte Volume Builder Module benötigt. Da DiPhAS Parameter Module bereits Knoten bei Slicer und Konverter bei OpenIGTLinkIF registriert, werden auch Status Knoten und Konverter von DiPhAS Parameter Module bei 3D Slicer und OpenIGTLinkIF registriert.

Zur Konfiguration des DiPhAS wird in der GUI von DiPhAS Parameter Module über das vtkSlicerNodeSelectorWidget „DiPhAS Preset“, das auch in Abbildung 27 zu sehen ist, ein vtkMRMLDiphasParameterModuleNode erzeugt. vtkSlicerNodeSelectorWidgets sind an einen entsprechenden Knotentyp gebunden, der über

```
this->diphasPresetNodeSelector->SetNodeClass(
        "vtkMRMLDiphasParameterModuleNode", NULL, NULL, "DiPhAS Preset");
```

definiert wird. “DiPhAS Preset” ist hierbei der angezeigte Namen des Knotens. Über vtkSlicerNodeSelectorWidgets können die Knoten des an sie gebundenen Knotentyps aus der Slicer Szene ausgewählt werden.

Um auf Events in Slicer zu reagieren, werden Observer benötigt. Wird einem Objekt ein Observer zugeordnet und diesem die zu überwachenden Events mitgeteilt, wird bei Auftreten dieses Events auf dem Objekt eine Methode aufgerufen, die dem Observer ebenfalls zuvor mitgeteilt werden muss. Um auf Gui Events zu reagieren, wird in Slicer „ProcessGUIEvents“ verwendet. Um auf MRML Events zu reagieren, wird „ProcessMRMLEvents“ verwendet. Für komplexe Module, die auch Events in der Logic verwenden, existiert außerdem ProcessLogicEvents.

In DiPhasParameterModule werden GUI Events überwacht. Wird ein Knoten über vtkSlicerNodeSelectorWidget ausgewählt, wird ein Pointer auf diesen erzeugt.

```
vtkMRMLDiphasParameterModuleNode* n =
        vtkMRMLDiphasParameterModuleNode::SafeDownCast(
                        this>diphasPresetNodeSelector->GetSelected());
```

Anschließend wird dieser im Modul als aktueller zu überwachender Knoten geladen.

```
vtkSetAndObserveMRMLNodeMacro( this->DiphasParameterModuleNode, n);
```

Die GUI-Objekte werden schließlich über UpdateGUI aktualisiert. Hierzu werden die Werte aus dem aktuell zu observierenden Knoten gelesen und den GUI-Objekten zugeordnet.

Erfolgt eine Änderung an einem GUI-Objekt, das einem Parameter wie z.B. Eindringtiefe zugeordnet ist, werden nach Aufruf von „ProcessGUIEvents“ durch den Observer die Änderungen in den aktuell zu observierenden Knoten übernommen und ein Modified Event auf diesem ausgelöst. Wurde in OpenIGTLinkIF ein Connector erzeugt, der mit einem externen Anwendungsteil, wie z.B. DiPhAS Networker verbunden ist, und wurde diesem der MRML Knoten zugeordnet, auf dem Änderungen in DiPhAS Parameter Module erfolgen, wird der Knoten nach Aufruf eines Modified Events von dem Nachrichtenkonverter vtkIGTLToMRMLDiphasControl in eine OpenIGTLink DiphasControl Nachricht übersetzt und an den externen Anwendungsteil versendet.

5.2.2. Volume Builder Module

Volume Builder Module dient der Nachbearbeitung und Rekonstruktion der 2D Ultraschalldatensätze. Volume Builder Module führt zu jedem Scan eine Liste, die Referenzen auf die einzelnen Ultraschallbilder, die zugehörigen Transformationen und die DiPhAS Konfigurationen enthält. Die referenzierten Objekte liegen als MRML Objekte in der aktuellen MRML Szene.

Abbildung 28 zeigt die Oberfläche von Volume Builder Module

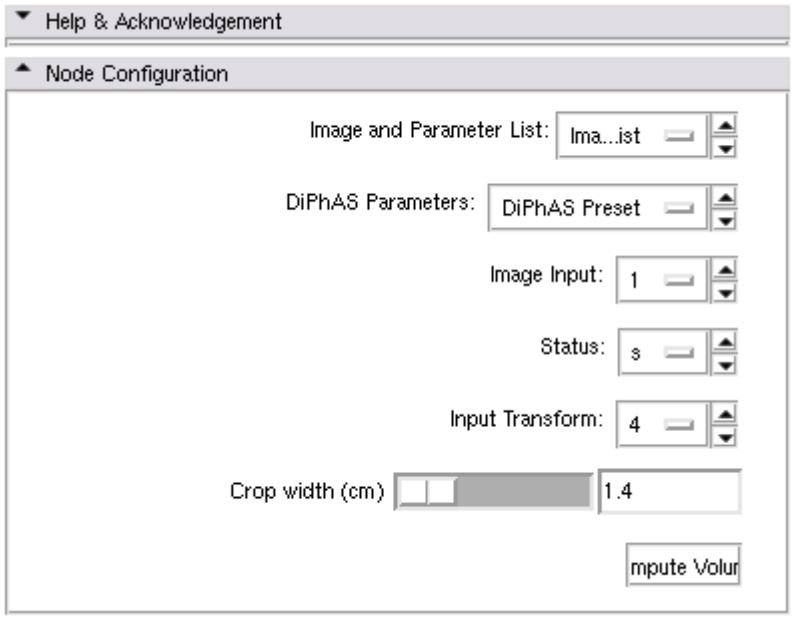

Abbildung 28: GUI zu Volume Builder Module

Listen müssen vom Benutzer von Hand erstellt werden. Vor jedem Scanvorgang muss eine neue Liste erstellt werden. Ein Scanvorgang beinhaltet hierbei das Scannen eines Objekts aus einer oder mehreren Richtungen. Die Algorithmen in Volume Builder Module, die in Abschnitt 5.4. beschrieben sind, erkennen verschiedene Orientierungen während eines Scanvorganges und berechnen die Datensätze entsprechend. Listen in Volume Builder Module sind als MRML Knoten realisiert.

Der zugehörige Knoten vtkMRMLVolumeBuilderModuleNode enthält hierzu als Liste einen std::vector. Dieser Vektor enthält wiederrum Vektoren von Character-Pointern, die auf die IDs von 2D-Ultraschallbild und DiPhAS Konfiguration zeigen. Ein Pointer auf die dem Bild zugeordnete Transformation ist nicht notwendig, da jeder vtkMRMLScalarVolume Node, in dem ein Bild automatisch von OpenIGTLinkIF beim Empfang abgelegt wird, eine Referenz auf einen vtkMRMLLinearTransformNode halten kann. Von OpenIGTLinkIF empfangene OpenIGTLink Transform-Nachrichten werden automatisch in vtkMRMLLinearTransform-Knoten abgelegt.

Um einen Scan mittels Volume Builder Module durchzuführen, muss eine Liste angelegt werden und dem Modul die verwendeten DiPhAS Parameter Knoten, Bildknoten, Transformationsknoten und Statusknoten bekannt gemacht werden.

Wichtig ist hierbei, dass zur Verwendung des DiPhAS Parameter Module Knoten und des Status Knoten das DiPhAS Parameter Module Projekt importiert sein muss. Damit ist gewährleistet, dass Volume Builder Module auf die entsprechenden Quellen zugreifen kann. Die Knoten müssen bei Initialisierung des Volume Builder Module bei Slicer registriert werden.

Beim Empfang eines Bildes oder einer Transformation wird diese angelegt und bei jedem weiteren Empfang eines Bildes oder einer Transformation von derselben Modalität aktualisiert. Sind Volume Builder die Transformations- und Bildknoten einmalig bekannt, kann auf neu eingetroffene Datensätze reagiert werden. Matlab ist für den Versand von Transformationen an 3D Slicer und der Status Nachricht an DiPhAS Networker zuständig. Damit kann garantiert werden, dass die Transformation an Slicer gesendet wird, bevor das Triggern (Versand der Status Nachricht „OK“) von DiPhAS Networker und damit der Versand des Ultraschallbildes an 3D Slicer erfolgt. Das bedeutet, dass beim Empfang eines Bildes von DiPhAS Networker durch Volume Builder Module die Transformation im observierten Transformations-Knoten, der zuvor durch die neue Transformation aktualisiert wurde, zu dem empfangen Bild gehört. Volume Builder Module kopiert bei jedem Empfang eines Bildknotens die observierten Transform-, Bild- und DiPhAS Konfigurationsknoten und fügt die Kopien der Slicer-Szene hinzu. Dem Bildknoten wird die Referenz des Transformationsknotens zugeordnet, so dass hier eine logische Verbindung entsteht. Dadurch ist 3D Slicer in der Lage, das 2D Bild an seiner korrekten Position im Raum darzustellen. Die Transformation wird später wieder benötigt, um den 3D Datensatz zu berechnen. Nachdem die Knoten der Slicer-Szene bekannt gemacht sind, kann die ID der Knoten innerhalb der Szene angefordert werden. Die IDs von Bild- und korrespondierendem DiPhAS Konfigurationsknoten werden hintereinander in einen Character-Vektor geschrieben und dieser der Liste (Vector im vtkMRMLVolumeBuilderModuleNode) für den Scan hinzugefügt.

Am Ende eines Scans existiert somit eine Liste, die Paare von Referenzen auf Bild- und Parameterknoten enthält. Mit diesen IDs können die Knoten wieder angefordert und weitergehende Verarbeitungsschritte durchgeführt werden.

Mittels eines Drucks auf „Compute Volume“ wird die Bildverarbeitungskette von Volume Builder Module gestartet.

Ultraschallbilder zeigen an der der Sonde zugewandten Seite helle Anteile, die aufgrund nicht vollständiger Ankopplung an das zu untersuchende Medium entstehen. Die dabei auftretenden Reflexionen werden als helle Anteile sichtbar. Zur Visualisierung werden Techniken des Volumenrenderings verwendet, die vor allem helle Anteile und damit starke Reflexionen darstellen. Die Ankopplungsartefakte sind hierbei aber nicht gewünscht. Deshalb enthält die Bildverarbeitungskette einen Algorithmus, der den der Sonde zugewandten Rand der Bilder abschneiden kann. Die Breite des Randes kann auf der GUI mit dem Schieberegler Crop Width in Zentimetern eingestellt werden.

Volume Builder erzeugt quaderförmige 3D Volumen. Für jede Orientierung der Sonde wird ein eigener Datensatz erzeugt. Bei Fahren einer linearen Trajektorie lassen sich die Bilder direkt zu einem quaderförmigen Datensatz kombinieren.

Der Roboter wird hierzu auf der Z-Achse der Sonde Verfahren und in äquidistanten Schritten auf der Z-Achse der Sonde gestoppt. Die aktuelle Transformation und das Bild werden an Slicer gesendet und die nächste Position angefahren.

Das Prinzip ist in Abbildung 29 verdeutlicht.

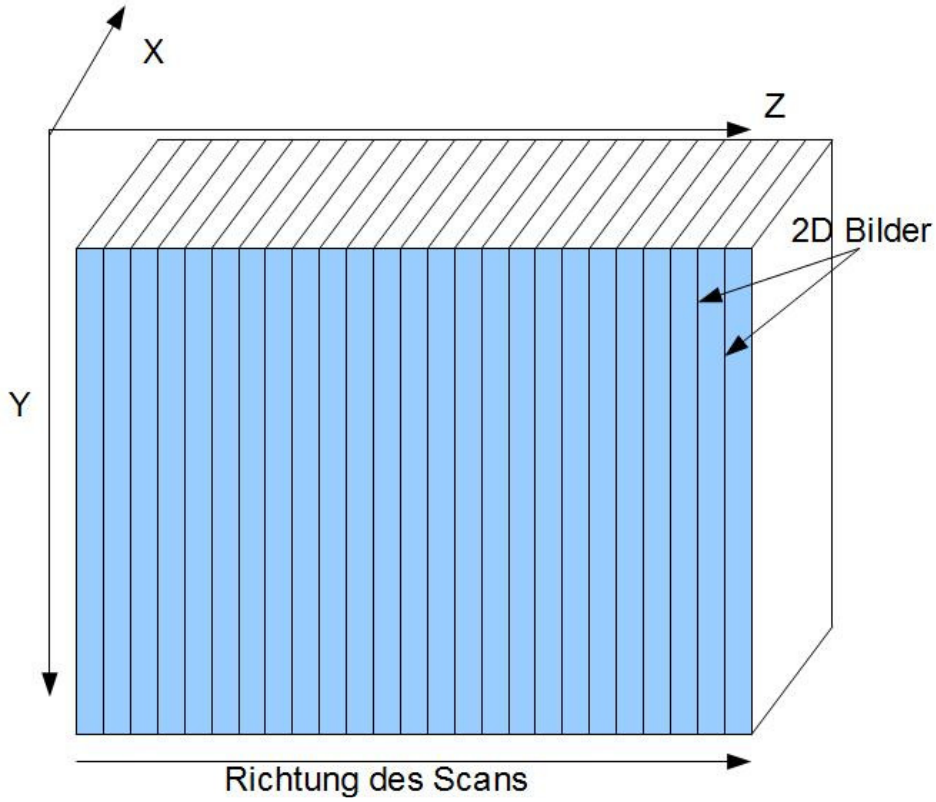

Abbildung 29: Volumen bei linearer Trajektorie

Wird die Sonde allerdings kraftgeführt entlang einer Oberfläche auf der Y-Achse der Sonde verschoben, sind auch die Bilder gegeneinander auf der Y-Achse verschoben. Die Bilder können nicht mehr direkt zu einem 3D Volumen kombiniert werden. Dies ist in Abbildung 30 zu sehen.

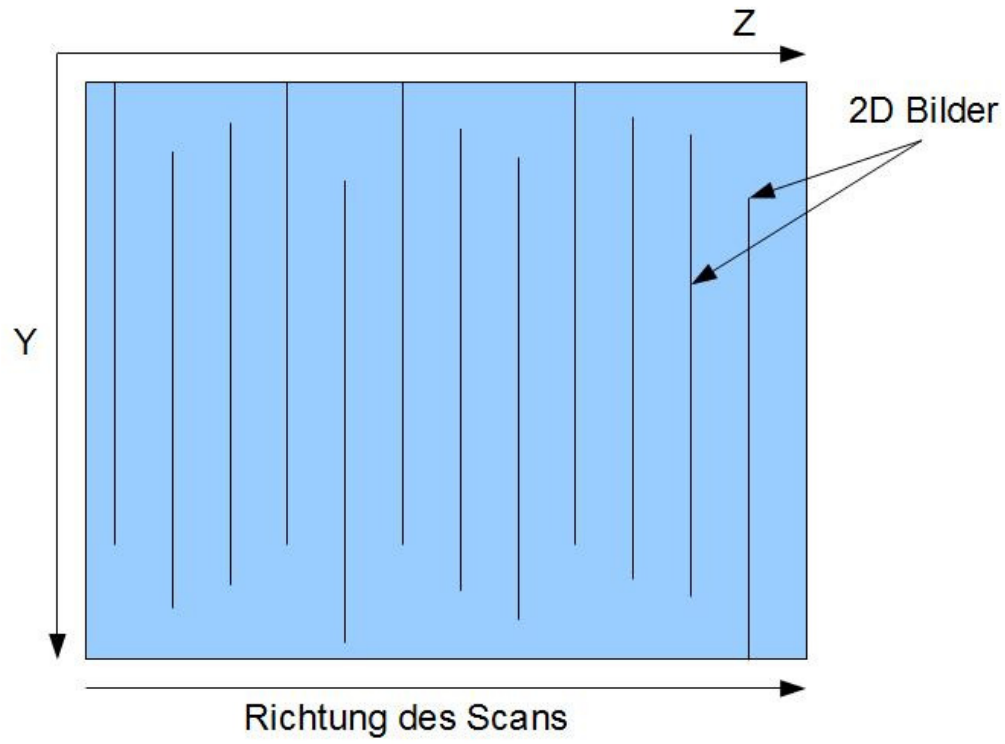

Abbildung 30: Volumen bei kraftgeführter Trajektorie

Die Bilder müssen damit sie direkt zu einem Quader kombiniert werden können, also nach oben bzw. unten aufgefüllt werden. Dies kann mit dem Grauwert 0 (Schwarz) erfolgen. Grauwert 0 tritt immer dort in Ultraschallbildern auf, wo keine Reflexion aufritt. Dort befinden sich also keine reflektierenden Objekte. Der Datensatz wird dadurch nicht verfälscht. Durch das Auffüllen nach oben und unten erhalten alle 2D-Datensätze gleiche Position auf der Y-Achse der Sonde und gleiche Ausdehnung in Y-Richtung des Bildes/der Sonde. Näheres hierzu ist Kapitel 5.4.3. zu entnehmen.

Nach dieser Berechnung werden im dritten Schritt der Bildverarbeitungskette die Volumen konstruiert. Dazu läuft ein Algorithmus durch die Liste, sucht alle Bilder, die gegeneinander auf der Sonden-X Achse keinen Versatz und identische Orientierung haben, berechnet für diese Bilder den Abstand zwischen den Bildern und erzeugt das 3D Volumen. Der Vorgang wird für jeden Versatz auf der X-Achse und jede Orientierung wiederholt. Für jede Orientierung und jeden Versatz auf der X-Achse der Sonde entstehen getrennte Datensätze, die gegeneinander automatisch registriert sind. Volume Builder kann diese Volumen mit unterschiedlichen Orientierungen nicht zu einem einzelnen Datensatz kombinieren. Durch die erfolgte Registrierung kann dies aber jederzeit in Nachverarbeitungsschritten erfolgen. Da Volume Builder Module keinen Versatz auf der X-Achse erlaubt, haben alle Datensätze die Breite (X-Achse) des Ultraschallarrays. Die Länge (Z-Achse) richtet sich nach der Länge der Trajektorie. Die Ausdehnung auf der Y-Achse (Höhe) wird bestimmt durch die Eindringtiefe und die Oberfläche, auf der sich die Sonde bewegt. Näheres hierzu ist Kapitel 5.4.4. zu entnehmen.

Die Verfahrrichtung und die Orientierung des Quaders verdeutlicht Abbildung 31.

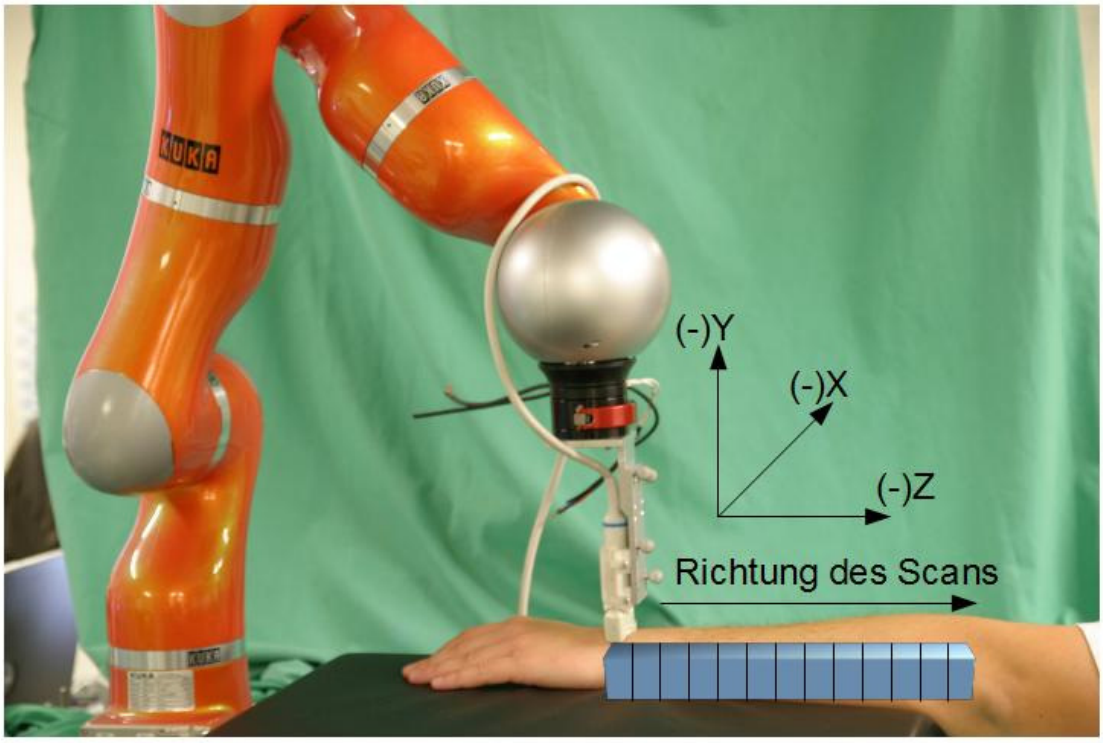

Abbildung 31: ungefähre Lage des Volumendatensatzes im Objekt. Die Oberfläche des Armes und die Vorzeichen des Koordinatensystems der Ultraschallsonde sind in dieser Abbildung nicht berücksichtigt

5.3. Steuerung der Roboter und Synchronisation der Modalitäten in Matlab

Matlab wird in dieser Arbeit zur Steuerung der Roboter und zur Synchronisation der Komponenten verwendet. Da zur Robotersteuerung sehr viele Basenwechsel notwendig waren, fiel die Entscheidung auf den Einsatz von Matlab. Matlab ist für die Arbeit mit Matrizen konzipiert. Deshalb können die notwendigen Berechnungen, die sehr viele Matrixoperationenen umfassen, sehr einfach in Skripte gefasst werden. Des Weiteren handelt es sich bei der Matlab Programmiersprache um eine Interpretersprache. Die Kommandos konnten also zeilenweise getestet und anschließend zu einem Skript kombiniert werden.

Mit dem in Matlab enthaltenen Oberflächendesigner GUIDE wurde die in Abbildung 32 abgebildete Matlab Oberfläche entwickelt.

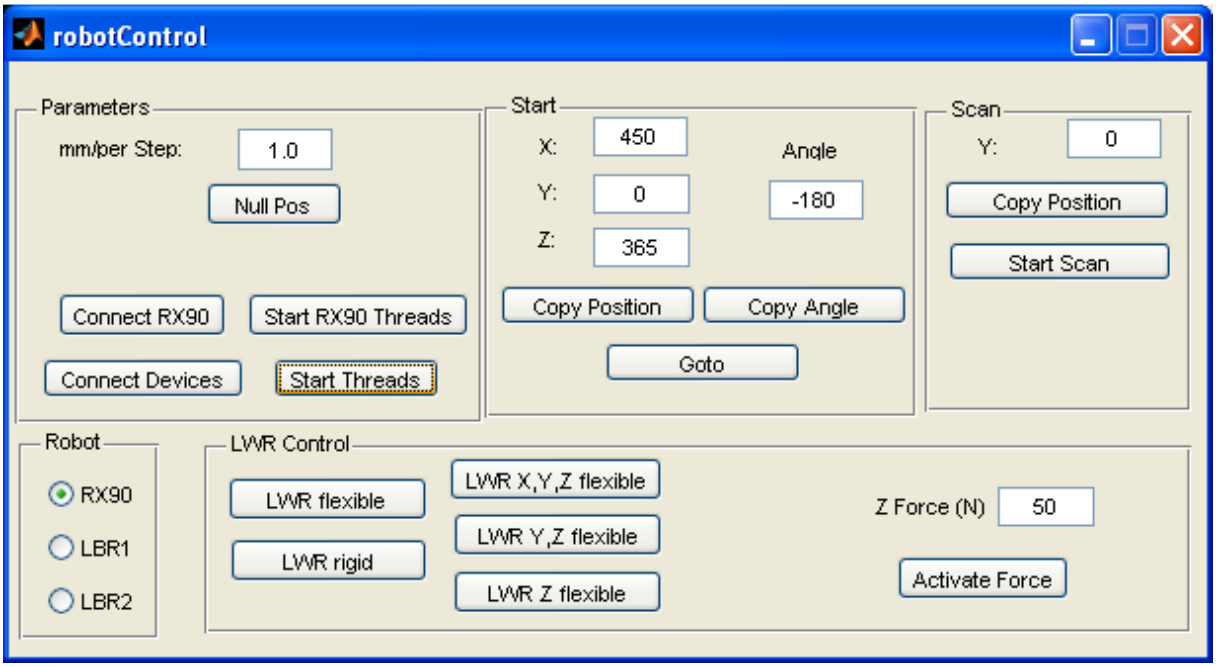

Abbildung 32: Oberfläche der Matlab Anwendung robotControl

Die einzelnen Bedienelemente rufen Matlab Funktionen in einem Matlab Skript auf. Jede Matlab Funktion besitzt einen eigenen Workspace. Variablen die in einer Funktion verwendet werden, sind in anderen Funktionen unbekannt. Um eine Variable funktionsübergreifend zu benutzen, muss diese in beiden Funktionen mit dem Schlüsselwort global markiert werden. Der hier verwendete Matlab Skript (im Folgenden robotControl) benötigt in mehreren Funktionen die Variable, die den verwendeten Roboter definiert und die OpenIGTLink Socket Deskriptoren, die eine offene OpenIGTLink Verbindung markieren.

„robotControl“ ist so entwickelt, dass alle Roboter verwendet werden können, die im Labor vorhanden sind. Der Nutzer kann im linken unteren Bereich des Anwendungsfensters den gewünschten Roboter wählen. Im Panel Parameter erfolgt die Konfiguration der Anwendung. Die Buttons Connect RX90 und Connect Devices öffnen die OpenIGTLink Verbindungen zu RX90 Roboter

bzw. den anderen Anwendungsteilen. Start RX90 Threads und Start Threads startet die notwendigen Threads, damit Matlab immer die aktuellsten Datensätze liest und nicht in einen Wartezustand verfällt (siehe Kapitel 5.1.3.). „mm/per Step“ definiert die Schrittweite für die Roboter. Wird mm/per Step auf 0.5 mm gesetzt, wird der Roboter auf einer linearen Trajektorie in 0.5 mm Abständen gestoppt und ein Bild akquiriert. Der Abstand der Slices im Volumen beträgt später ebenfalls 0.5 mm. Mit dem Button Null Pos kann der aktuell gewählte Roboter an eine sichere fest definierte Ausgangsposition gefahren werden.

Das Panel „LWR Control“ dient zur Konfiguration der Roboter bei Verwendung eines der beiden LWR IV Roboter. „LWR flexible“ setzt die Steifigkeit aller Achsen auf den Minimalwert. Der Roboter lässt sich dann über seine Kraftmomentensensoren frei im kartesischen Raum positionieren. „LWR rigid“ setzt die Steifigkeit auf den Maximalwert. Der Roboter lässt sich auf keiner Achse durch externe Kräfte positionieren und hält sowohl Position als auch Orientierung. Im Rigid Modus kann ein LWR ähnlich wie ein RX90 verwendet werden. Die Genauigkeit beim Einsatz des LWR ist im Gegensatz zum RX90 allerdings reduziert (siehe Kapitel 4.6. und 4.7.).

Zur Verdeutlichung der folgenden Abschnitte zeigt Abbildung 33 das Koordinatensystem des Roboterendeffektors

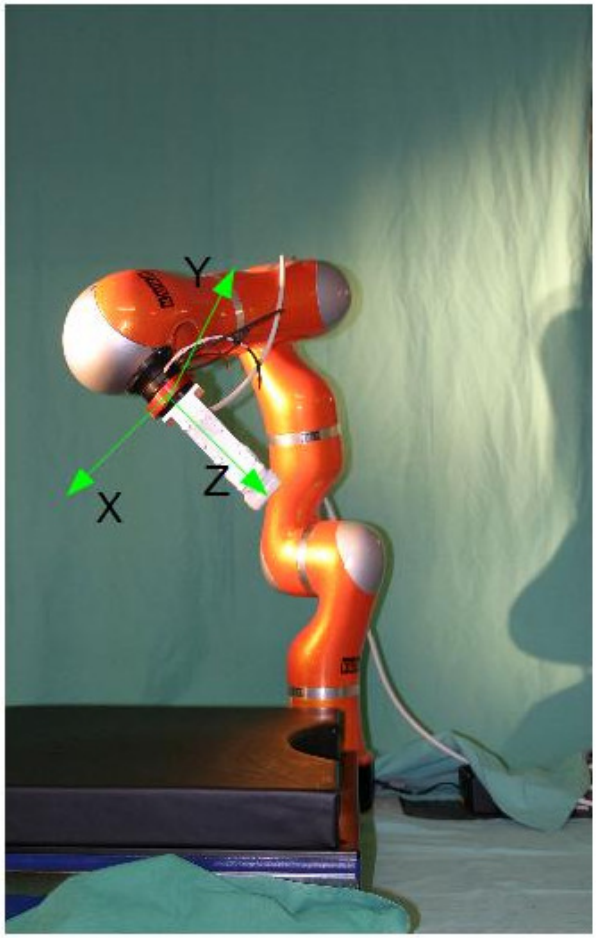

Abbildung 33: Endeffektorkoordinatensystem des Roboters

„LWR X, Y, Z flexible“ setzt die Steifigkeit des Roboters für X, Y, Z Achse im kartesischen Koordinatensystem des Endeffektors auf den Minimalwert. Die Steifigkeit für die Orientierung des Endeffektors wird auf den Maximalwert gesetzt. So kann die Position des Endeffektors frei gewählt werden, die Orientierung jedoch nicht. „LWR Y, Z flexible“ setzt zusätzlich den Steifigkeitswert der X-Achse auf den Maximalwert, so dass der Endeffektor nur noch auf seiner Y und Z-Achse frei bewegt werden kann. „LWR Z flexible schließlich setzt alle Steifigkeiten, bis auf die Steifigkeit der Z-Achse auf Maximum. Die Steifigkeit für die Z-Achse wird auf den Minimalwert gesetzt. Der Roboter ist nach Druck des Buttons nur noch auf der Z-Achse des Endeffektors durch externe Kräfte frei bewegbar. Der „LWR Z flexible“ Modus wird zum Folgen der Oberfläche verwendet. X und Y Achse, sowie Orientierung des Endeffektors sind fixiert. Gibt der Benutzer dem Roboter nun eine Position für X und Y Achse an, so fährt der Roboter diese Position an. Wirkt auf den Roboterflansch (Z-Achse) ausschließlich eine Kraft, die der Gewichtskraft entspricht, die aus dem Gewicht des Endeffektors und der Schwerkraft resultiert, so wird die Position auf der Z-Achse gehalten, da die Steifigkeit auf der Z-Achse in diesem Modus auf den Minimalwert gesetzt wurde und die Schwerkraft ausgeglichen wird. Der Roboter wird auf der Z-Achse eine Position einnehmen, in der ein Gleichgewichtszustand hergestellt ist. Dies können alle Positionen auf der Z-Achse des Endeffektors sein, in denen ausschließlich die Gewichtskraft auf den Endeffektor wirkt.

Während des Scans kann so aber nicht dem Gewebe gefolgt werden, da die Gleichgewichtsposition an jedem Punkt im Arbeitsraum hergestellt ist, an dem der Roboter keine Objekte berührt. Der Benutzer kann nun auf die Z-Achse des Endeffektors mit einer Hand Druck ausüben. Der Roboter wird dieser Kraft folgen. Auf diese Weise kann der Endeffektor auf dem zu untersuchenden Objekt aufgelegt und durch äußere Zuführung von Kraft auf der Z-Achse diesem auch während des Verfahrens auf X- bzw. Y-Achse gefolgt werden. Mit dieser Methode kann die Auflagekraft allerdings nicht konstant gehalten werden. Die Kuka LWR Steuerung bietet die Möglichkeit, auf einer beliebigen Achse eine Kraft anzulegen. Die gewünschte Kraft für die Z-Achse kann im Feld Z Force(N) von robotControl in Newton angegeben werden. Nach einem Druck auf Activate Force wird der Roboter, sofern er sich gerade in einem Modus befindet, in dem die Steifigkeit der Z-Achse des Endeffektors minimal ist, in Richtung der positiven Z-Achse des Endeffektors fahren bis die gewünschte Kraft am Endeffektor anliegt. Diese Kraft wird nun bis zum Beenden dieses Modus gehalten. Wird der Roboter nun auf X- oder Y-Achse des Endeffektors verfahren, so folgt der Endeffektor durch Halten der Kraft auf der Z-Achse der Oberfläche der Objekte in Verlängerung der montierten Ultraschallsonde. Der RX90 unterstützt diese Funktion mangels eingebauter Kraftmomentensensorik nicht und kann in dieser Anwendung nur lineare Trajektorien fahren.

Das Panel Start legt die Startposition für einen Scanvorgang fest. Die Startposition kann in kartesischen Koordinaten im Roboterbasiskoordinatensystem von Hand eingegeben werden.

Aus Gründen der Vereinfachung kann eine Trajektorie nur in der Y, Z-Ebene des Roboterbasiskoordinatensystems gefahren werden. Die X-Koordinate muss an allen Positionen während des Verlaufs des Scans konstant gehalten werden. Die Sonde zeichnet dann an jeder Halteposition ein Bild in der X, Z-Ebene des Roboterbasiskoordinatensystems auf. Damit alle Slices in den Serien parallel zueinander liegen, ist die Rotation der Sonde bei dieser Aufnahmetechnik nur um die Y-Achse des Roboterkoordinatensystems sinnvoll.

In Abbildung 34 ist das Prinzip verdeutlicht.

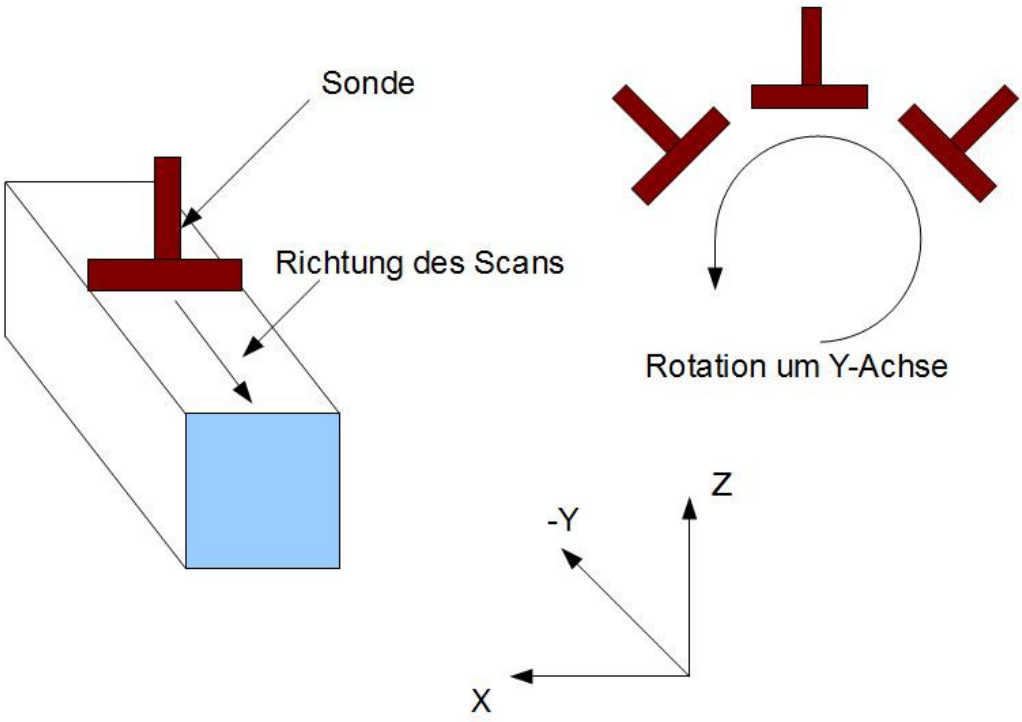

Abbildung 34: Basiskoordinatensystem und Rotation der Sonde

Im Feld Angle kann der Winkel für die Rotation der Sonde eingegeben werden. Das Koordinatensystem der Ultraschallsonde entspricht nicht dem Koordinatensystem der Roboterbasis. Die X-Achse der Roboterbasis entspricht der negativen Z-Achse der Sonde. Die Y-Achse der Roboterbasis entspricht der X-Achse der Sonde. Die Z-Achse der Roboterbasis entspricht der Y-Achse der Sonde.

Es ergibt sich folgende Transformation für den Übergang aus Roboterbasiskoordinaten in Sondenkoordinaten:

$$\begin{pmatrix} 0 & 0 & -1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}$$

Formel 13: Transformation für den Übergang von Roboterbasiskoordinaten in das Koordinatensystem der Ultraschallsonde

Abbildung 35 illustriert die Transformation zwischen Sonde und Roboter.

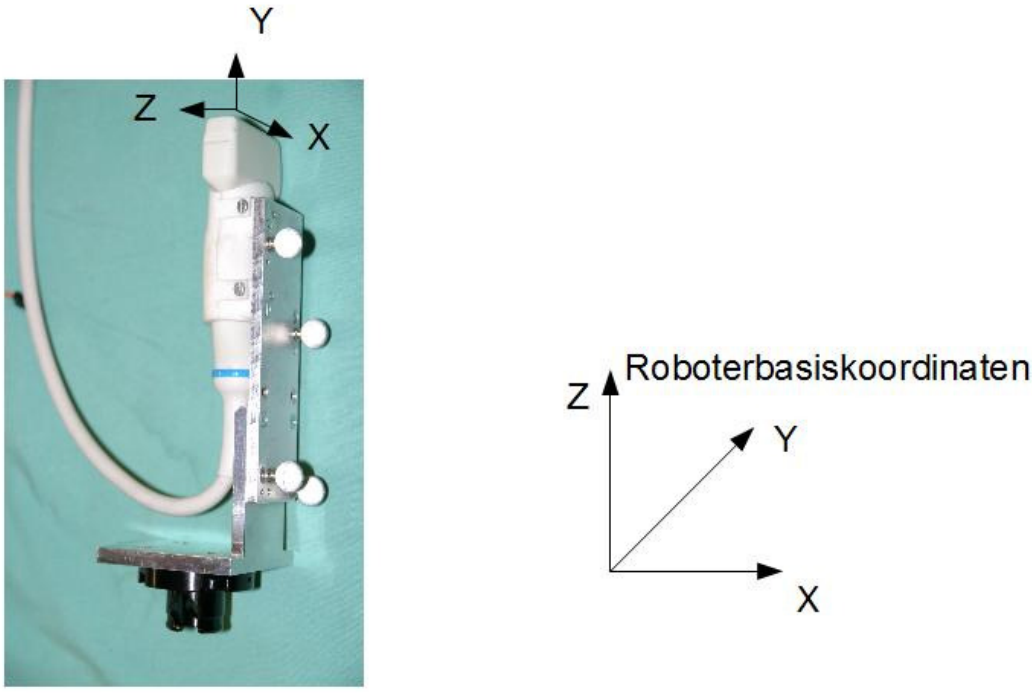

Abbildung 35: Koordinatensysteme von Sonde und Roboterbasis

Da Die Z-Achse der Roboterbasis bei Verwendung der Einheitsmatrix (also ohne Rotation) senkrecht nach oben zeigt muss ein Winkel von 180 Grad verwendet werden, damit die Y-Achse der Sonde (Z-Achse des Endeffektors) in Richtung der negativen Z-Achse der Roboterbasis zeigt (die Sonde zeigt senkrecht nach unten). Die Bildkoordinaten entsprechen den Koordinaten der Ultraschallsonde.

Aufgrund der Montagerichtung ergibt sich für die Rotation der Sonde um die Z-Achse der Roboterbasis ein Winkel von -90 Grad, damit beim Durchführen eines Scans und beim Verfahren auf der Y-Achse in Roboterbasiskoordinaten die Sonde sich auf ihrer negativen Z-Achse bewegt und damit Bilder in der X, Z Ebene in Roboterbasiskoordinaten akquiriert, die die Slices in Y-Richtung im Volumendatensatz bilden.

Abbildung 36 zeigt die Orientierung der Sonde, wenn eine Rotation von -90 Grad um die Z-Achse der Roboterbasis und eine Rotation von 180 Grad um die Y-Achse der Roboterbasis durchgeführt werden.

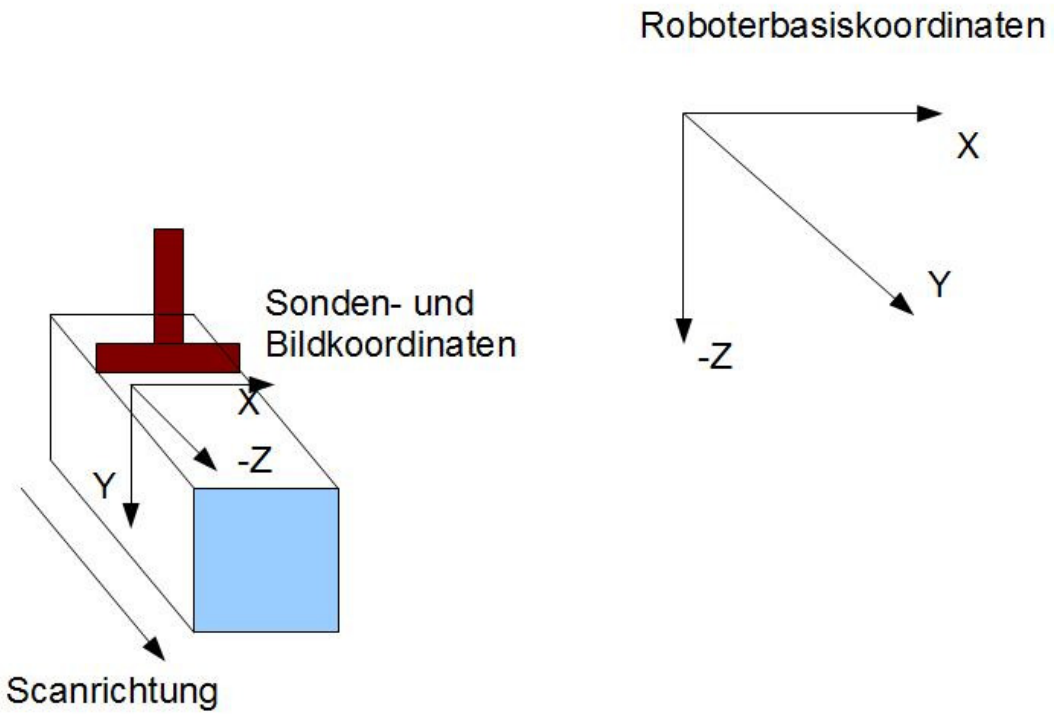

Abbildung 36: Orientierung der Sonde bei Drehung von -90 Grad um die Z-Achse der Roboterbasis und 180 Grad um die Y-Achse der Roboterbasis

Die Startposition in der GUI von robotControl gibt einen Punkt in Verlängerung des Roboterflansches in Basiskoordinaten an, der auf der Achse der ersten Fokussierung liegt. Dieser Punkt ist der Toolcenterpoint (TCP) für diese Anwendung. Der Punkt wurde für den Fall gewählt, dass ein Objekt aus verschiedenen Orientierungen gescannt werden soll. Soll um eine Achse in einem Objekt gedreht werden, sind in der Regel die Eigenschaften der Objekte von Interesse, die sich auf dieser Achse befinden. Dort soll eine möglichst große Genauigkeit erreicht werden. Deshalb wird die Achse und damit die Tiefe für die erste Fokussierung so gesetzt, dass sie die Achse schneidet, um die rotiert werden soll. Die Lage der Rotationsachse im Objekt ergibt sich durch Verschieben des Toolcenterpoints auf der Y-Achse in Roboterbasiskoordinaten (Achse entlang der Trajektorie)

Abbildung 37 verdeutlicht die Lage des Toolcenterpoints im Ultraschallbild und im Raum.

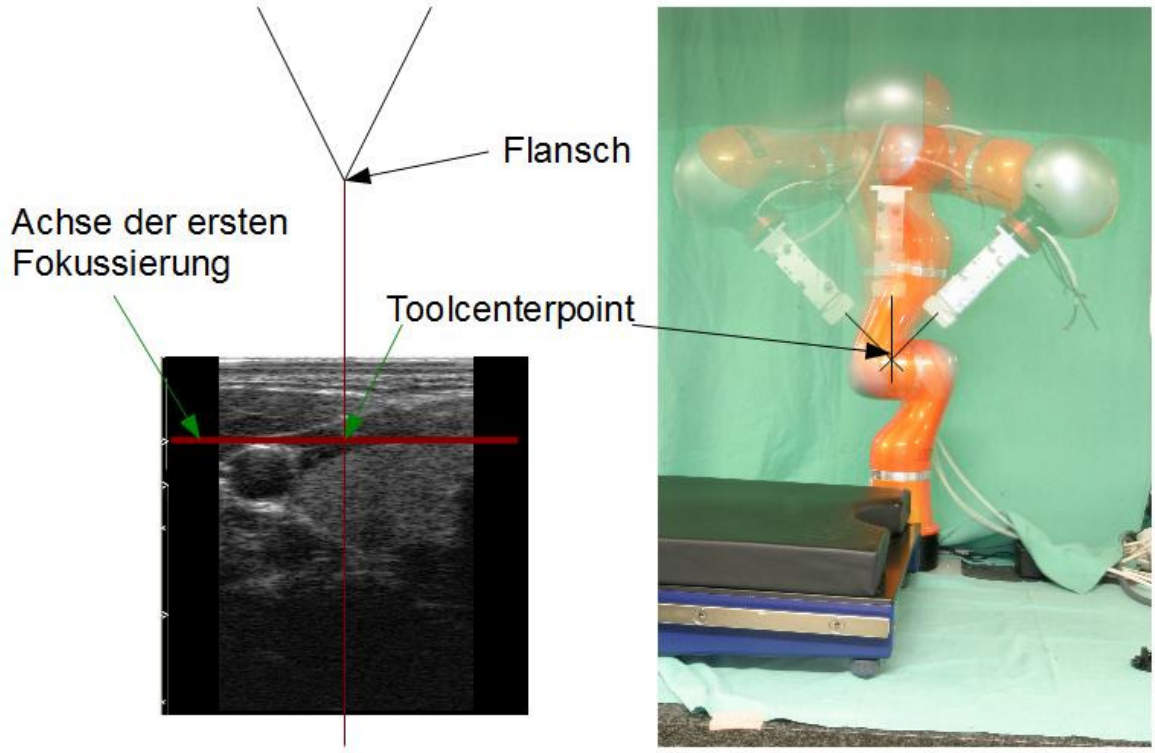

Abbildung 37: Lage des Toolcenterpoints im Ultraschallbild und im Raum

Ein Druck auf Copy Angle liefert den aktuellen Orientierungswinkel der Sonde um die Y-Achse der Roboterbasis. Ein Druck auf Copy Position berechnet die Position des Toolcenterpoints bei Verwendung des aktuell eingestellten Winkels.

Um diesen Punkt zu berechnen, benötigt die Anwendung Kenntnis über die Abmessungen des Werkzeuges und den aktuell verwendeten Parametersatz von DiPhAS. Die Abmessungen des Tools wurden einmalig vermessen. Diese Transformation ist in der Anwendung hinterlegt und kann jederzeit abgefragt werden. Aufgrund der Verwendung des Eventsystems von Slicer wird bei jeder Änderung der Parameter des DiPhAS der aktuelle Parametersatz auch an die Matlab Anwendung gesendet, sofern OpenIGTLinkIF entsprechend konfiguriert ist.

Um den TCP zu berechnen wird die Position des Roboterflansches von Basiskoordinaten in Sondenkoordinaten transformiert. Anschließend genügt es die Translation für das Werkzeug (Abstände von Werkzeugspitze zum Roboterflansch) zur Translation des Roboterflansches in Sondenkoordinaten zu addieren. Der berechnete Punkt befindet sich im Zentrum der Auflagefläche (auf dem Gewebe) der Ultraschallsonde. Um schließlich den TCP zu berechnen muss zur Translation außerdem auf der Z-Achse in Roboterflanschkoordinaten oder der Y-Achse in Sondenkoordinaten die Tiefe des ersten Fokus addiert werden.

Dadurch ist der Startpunkt für den Scan definiert. Dieser kann im „LWR flexible“ Modus durch Bewegen des LWR durch externe Kräfte und eines Drucks auf Copy Position ausgemessen werden.

Ein Druck auf Goto verfährt den Roboter so, dass der TCP die gewünschte Position einnimmt. Um einen Scan durchzuführen muss nur noch eine Zielposition angegeben werden. Der Roboter bewegt sich während des Scans nicht auf der X-Achse des Sondenkoordinatensystems. Die Z-Position ist bei linearen Trajektorien ebenfalls fix, bzw. wird bei kraftgeführten Trajektorien zur Laufzeit des Scanvorgangs auf Basis der Kräfte berechnet. Deshalb muss nur noch die Y-Komponente der Zielposition für den TCP angegeben werden. Auch diese Position kann aus der aktuellen Roboterposition mittels Copy Position im Scan Panel berechnet werden. Ein Druck auf Start Scan startet den Scan Vorgang. Der Roboter wird von der Start- an die Zielposition gefahren und auf der Trajektorie in äquidistanten Schritten auf der Y-Achse (Roboterbasiskoordinaten) unter Verwendung der in mm/per Step angegeben Schrittweite gestoppt. Hier kann je nach verwendetem Roboter entweder eine lineare Trajektorie oder eine kraftgeführte Trajektorie verwendet werden. An jeder Position, an der der Roboter gestoppt wird, wird die aktuelle Robotertransformation zur Berechnung der Bildkoordinaten herangezogen. Diese Bildkoordinaten entsprechen den Sondenkoordinaten und müssen zur Generierung und Registrierung der Volumendatensätze an Slicer gesendet werden. Die Berechnung erfolgt durch Wechseln der Basis ins Koordinatensystem der Ultraschallsonde. Eine Subtraktion der ersten Fokustiefe auf der Y-Achse in Sondenkoordinaten liefert den Punkt im Zentrum der Auflagefläche der Ultraschallsonde. Die homogenen Koordinaten dieses Punktes werden in Roboterbasiskoordinaten an Slicer übertragen und dort von Volume Builder verwendet. Anschließend erfolgt das Triggern von DiPhAS Networker, damit dieser ein Bild an Slicer sendet. Matlab wartet solange bis Volume Builder das Bild empfangen hat und seine „OK“ Status Message an Matlab versendet. Anschließend berechnet Matlab die nächste Position auf der Trajektorie und fährt diese an. Ist die Zielposition erreicht, ist der Scanvorgang beendet.

Um zu gewährleisten, dass der Roboter die gewünschte Position erreicht hat bevor die Transformation vom Roboter angefordert und an Slicer gesendet wird, pausiert Matlab solange, bis der Roboter die Position sicher erreicht hat.

5.4. Algorithmen

Kapitel 5.4 beschreibt die von DiPhAS Networker und Volume Builder verwendeten Algorithmen, die während dieser Arbeit entwickelt wurden im Detail.

5.4.1. Algorithmus zum Entfernen der Bildränder

DiPhAS Networker beinhaltet einen Algorithmus zur Vorverarbeitung der 2D Ultraschallbilder. DiPhAS liefert B-Mode Bilder als RGB Datensätze, obwohl diese nur Grauwertinformationen enthalten. Die Kanäle für rot, grün und blau eines Pixels enthalten deshalb jeweils den gleichen Wert. Für jedes Pixel werden in einem ersten Schritt zwei der drei Kanäle verworfen. Die Grauwerte der einzelnen Pixel werden zeilenweise durchlaufen und in einem Array abgelegt. Für ein 600x600 Pixel großes Bild ergeben sich so 36000 Werte. Die 600x600 Pixel großen Bilder enthalten je nach gewählten Parametern Anteile, die keine Nutzinformationen tragen (siehe Kapitel 5.1.4). Diese Anteile enthalten lediglich den Grauwert schwarz (0) und werden im nächsten Schritt entfernt. DiPhAS berechnet die Bildinformationen so, dass keine Verzerrung in den Bildern vorhanden ist. Die Fläche im Raum, die ein Pixel repräsentiert, muss also quadratische Ausdehnung haben. Der gesamte Raum eines quadratischen Bildes wird deshalb nur genutzt, wenn die Eindringtiefe der Breite der Sonde entspricht. Ist die Eindringtiefe geringer, werden schwarze Anteile am unteren Bildrand eingefügt. Ist die Eindringtiefe größer als die Breite der Sonde, wird die Nutzinformation zentriert und schwarze Anteile werden links und rechts der Nutzinformation eingeblendet.

Da das DiPhAS API einen Fehler enthält, der das Ausblenden der Labels für Eindringtiefe und Fokussierungen, die sich am linken Bildrand befinden verhindert, muss darauf geachtet werden, dass die Eindringtiefe größer gewählt wird als die Breite der Sonde. Wird die Eindringtiefe so gewählt, dass die Labels vollkommen im schwarzen Bereich liegen, kann der im Folgenden beschriebene Algorithmus sie entfernen. Um die schwarzen Anteile zu entfernen und nur die Nutzinformation zu übertragen, werden die Ankopplungsartefakte benutzt. Am oberen Rand eines Bildes entstehen immer mehr oder weniger stark ausgeprägte Ankopplungsartefakte. Diese Artefakte korrespondieren mit hellen Bildanteilen. Der Algorithmus beginnt in der rechten oberen Ecke des Ultraschallbildes und durchläuft eine der ersten Zeilen so lange von rechts nach links bis er auf einen Pixel stößt, dem ein von 0 verschiedener Grauwert zugeordnet ist. Da in den ersten Zeilen eines Ultraschallbildes davon ausgegangen werden kann, dass Ankopplungsartefakte vorhanden sind, ist dann der rechts der Mitte liegende Rand der Nutzinformation erreicht, wenn der erste von 0 verschiedene Grauwert gefunden wurde. Die Anzahl der Pixel vom rechten Nutzbildrand bis zum Rand des 600x600 Pixel großen Bildes wird gezählt. Ist dieser Bereich z.B. 200 Pixel breit, ist auch der Bereich auf der linken Seite des Ultraschallbildes 200 Pixel breit, da die Nutzinformation zentriert ist. Um nur die Nutzinformationen zu übertragen, müssen für jede Zeile die Pixel 0-199 und 400-599 verworfen werden.Erwünschter Nebeneffekt ist die Entfernung der Labels, die sich links des Ultraschallbildes in diesem Bereich befinden. Die Restbreite des Bildes beträgt im Beispiel 200 Pixel. Es wird also ein Array mit 200x600 Werten erstellt und für jede Zeile des Ultraschallbildes die Werte 200-399 in dieses Array geschrieben. Übrig bleibt nur die Nutzinformation, die anschließend via OpenIGTLink übertragen werden kann. Da sich die Breite des Bildes im Betrieb nur ändert, wenn neue Parameter empfangen wurden, muss die Berechnung der Breite der Ränder nur durchgeführt werden, wenn eine Rekonfiguration erfolgt. Der Quellcode des Algorithmus ist in Anhang A hinterlegt.

5.4.2. Algorithmus zum Entfernen der Ankopplungsartefakte

Die für den Algorithmus aus 5.3.1. nützlichen Ankopplungsartefakte sind bei der Volumenvisualisierung mittels Volume Rendering unerwünscht. Bei der Verwendung von Compositing oder der Maximum Intensity Projection, die beide während dieser Arbeit zum Einsatz kommen, werden helle Bildanteile, die mit Reflexionen in Ultraschallbilder korrespondieren und damit Grenzflächen darstellen, als Oberfläche dargestellt. Damit wird die Rekonstruktion von Objekten innerhalb des Volumens ermöglicht. Die Ankopplungsartefakte sind in Ultraschallbildern ebenfalls als helle Bildanteile sichtbar. Zur Volumenvisualisierung sind sie unerwünscht, da sie beim Compositing, oder der Maximum Intensity Projection ebenfalls als Objekte dargestellt werden. Um diese Artefakte zu entfernen, soll bewusst auf die Anteile der Ultraschallbilder, in denen auch die Artefakte liegen, verzichtet werden. Diese Bereiche sollen mit dem Grauwert 0 (schwarz) gefüllt werden, der mit keinerlei Reflexion korrespondiert.

Die Entfernung dieser Bildanteile, die sich in den ersten Zeilen eines Ultraschallbildes befinden, erfolgt im 3D Slicer Modul Volume Builder. Der Nutzer kann für eine Liste von Ultraschallbildern, Transformationen und Parametersätzen, die bereits gesammelt wurden, angeben wie hoch (in cm) der Anteil der Bilder sein soll, der geschwärzt wird. Dieser Wert wird in Zentimetern angegeben. Das Ausführen der Bildverarbeitungskette von Volume Builder bewirkt, dass für jedes Bild in der Liste die ersten n Zeilen geschwärzt werden. Die Anzahl n der zu schwärzenden Zeilen wird aus der Eindringtiefe aus der Konfiguration für das Ultraschallgerät zum Zeitpunkt der Akquisition des aktuellen Bildes, aus der Breite des Randes (in cm) und aus den Abmessungen des Bildes berechnet. Hierzu wird die Eindringtiefe durch die Anzahl der Pixel des Bildes in Schallrichtung geteilt. Der berechnete Wert ist die Ausdehnung eines Pixels in Schallausbreitungsrichtung. Anschließend wird die Höhe des zu schwärzenden Randes durch die Ausdehnung eines Pixels geteilt. Das Ergebnis ist die Anzahl der zu schwärzenden Zeilen.

Um den schwarzen Rand in ein Bild einzufügen wird ein Array ausreichender Größe mit dem Grauwert 0 (schwarz) gefüllt. Der Speicher für dieses Array wird mit malloc dynamisch allokiert. Anschließend wird ein Pointer auf das Array erstellt, das die Bilddaten des Ultraschallbildes enthält. Da immer die ersten n Zeilen eines Bildes manipuliert werden müssen, ist der Offset im Array 0 und die Anzahl der zu schwärzenden Pixel n* “Anzahl der Pixel in einer Bildzeile“. Schließlich werden mittels memcpy n* “Anzahl der Pixel einer Bildzeile“ aus dem Array, das ausschließlich den Grauwert 0 enthält, an den Anfang des Arrays mit den Bildinformationen kopiert. Die ersten n Zeilen sind somit auf Grauwert 0 (schwarz) gesetzt. Dieser Vorgang wird für jedes Bild in der Liste wiederholt.

Der Quellcode des Algorithmus ist in Anhang B hinterlegt.

5.4.3. Algorithmus zum Einpassen der Bilder in Volumendatensätze

Im nächsten Schritt der Bildverarbeitungskette müssen die Bilder wie in Kapitel 5.2.2. beschrieben in einen quaderförmigen Raum eingepasst werden. Um die Bilder in einen ebenfalls quaderförmigen Volumendatensatz schreiben zu können, müssen alle Transformationen für die einzelnen Bilder dieselbe Rotation enthalten. Die Bilder sollen als Slices hintereinander in Richtung der negativen Z-Achse der Sonde in einen Volumendatensatz geschrieben werden. Deshalb müssen für alle Bilder zusätzlich zur Rotation auch die Translationen auf X- und Y-Achse identisch sein. Die Differenz der Y-Komponenten der Translationen zweier aufeinanderfolgender Bilder gibt den Abstand der Slices im Volumen an. Die einzelnen Slices im Volumen sind nach dem Einpassen in den Volumendatensatz nicht fest definiert, sondern können mittels multiplanarer Rekonstruktion in beliebiger Orientierung aus dem Volumendatensatz erzeugt werden.

Die Transformationen die einzelne Bilder haben müssen, um in einen Volumendatensatz mit der Transformation

$$\begin{pmatrix} 1 & 0 & 0 & 10 \\ 0 & 1 & 0 & 10 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

und mit 1mm Abstand der Pixel auf der Z-Achse integriert werden zu können, sind in Abbildung 38 illustriert.

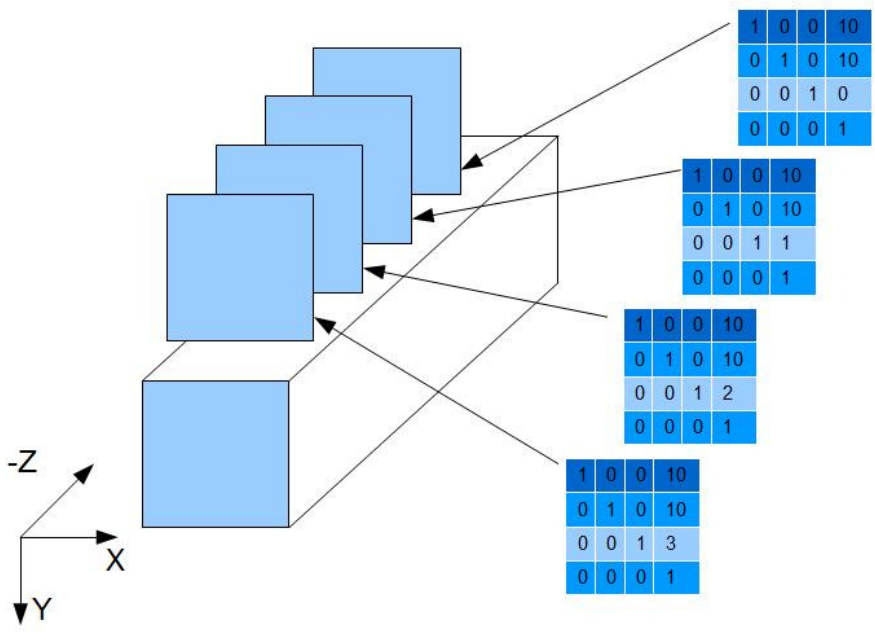

Abbildung 38: Transformationen der einzelnen Ultraschallbilder

Da die Bilder allerdings wie bereits in Kapitel 5.2.2. Abbildung 30 zu sehen bei Verwendung einer kraftbasierten Trajektorie unterschiedliche Translationen auf der Y-Achse (Z-Achse des Endeffektors) haben, können sie nicht direkt zu einem Volumendatensatz kombiniert werden. Die Differenzen der Y-Komponenten der Translationen ergeben sich hierbei durch die kraftbasierte Verfolgung der Oberfläche. Die Verfolgung der Oberfläche wird ausschließlich durch die kraftbasierte Führung des Roboters auf der Y-Achse der Sonde realisiert. Dies wird in Abbildung 31 ersichtlich. Um dennoch pro Trajektorie, bei der sich die Orientierung der Sonde und Translation auf der X-Achse der Sonde nicht ändern, einen einzigen Volumendatensatz erzeugen zu können, müssen die Bilder modifiziert werden.

Abbildung 39 zeigt ein Beispiel für Ultraschallbilder mit unterschiedlichen Translationen auf der Y-Achse.

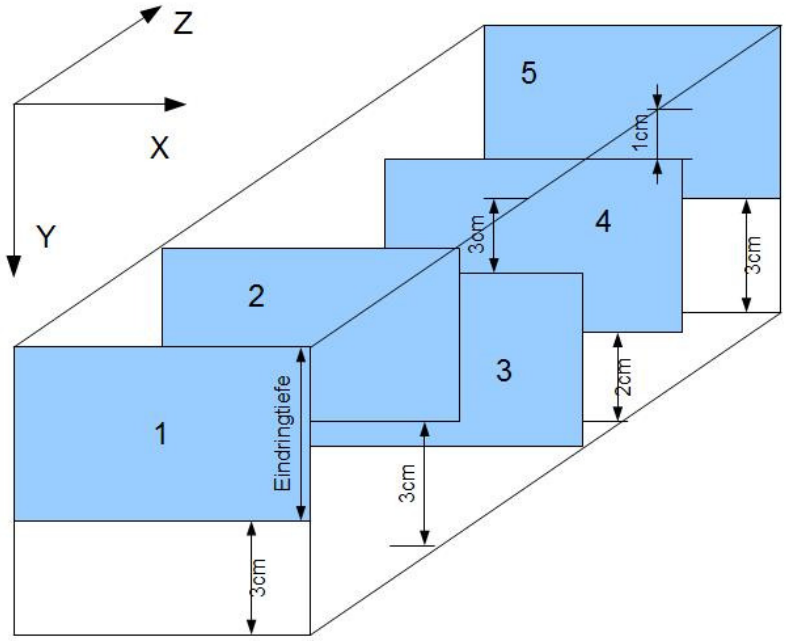

Abbildung 39: Unterschiedliche Lage der Nutzinformationen in einem quaderförmigen Raum

Das Verfahren wird im Folgenden am Beispiel in Abbildung 39 erläutert. Damit ein korrekter Datensatz erstellt werden kann, muss jedes Slice dieselbe Ausdehnung auf seiner X- und Y-Achse haben. Um die Ultraschallbilder an ihrer korrekten Position im Volumendatensatz positionieren zu können, muss zuerst die Höhe des späteren Volumendatensatzes berechnet werden. Hierzu müssen das Slice mit dem kleinsten Wert für die Translation auf der Y-Achse und das Slice mit dem größten Wert für die Translation auf der Y-Achse gesucht werden. Im Beispiel sind das die Slices 1 und 3. Die Ausdehnung des Volumendatensatzes und damit der einzelnen Slices auf der Y-Achse berechnet sich aus dem Betrag der Distanz zwischen den Y-Komponenten der Translationen der beiden Slices und der Eindringtiefe für die aktuelle Trajektorie. Die Summe aus Eindringtiefe und dem Betrag ergibt die Ausdehnung der einzelnen Slices bzw. des Volumendatensatzes auf der Y-Achse in Zentimetern. Für jedes Slice wird ein mit Grauwert 0 (schwarz) gefülltes Array mit der Größe von g*h Elementen

erstellt, wobei g die neue Ausdehnung auf der Y-Achse und h die Ausdehnung auf der X-Achse in Pixeln ist. Die Ausdehnung in Zentimeter der einzelnen Pixel in X-, Y- und Z-Richtung ist durch die Spacings eines Bildes bekannt. Mit diesen Spacings kann der Zusammenhang zwischen Bildgröße und Anzahl der Pixel hergestellt werden und damit zwischen Pixeln und Zentimetern umgerechnet werden. Anschließend wird für jedes Bild der Serie die Distanz zwischen der Y-Komponente seiner Translation und der Y-Komponente der Translation des Bildes mit dem kleinsten Wert der Y-Komponente der Translation (im Beispiel Slice 1) berechnet. Mittels der Pixelgröße (Spacings) wird berechnet, wie vielen Bildzeilen dieser Wert entspricht. Die Anzahl der Bildzeilen(n) multipliziert mit der Ausdehnung eines Bildes auf der X-Achse (m) ergibt den Offset, der im letzten Schritt für memcpy verwendet wird. Mittels memcpy wird die Nutzinformation in das für das Bild neu erstellte Array, das mit Grauwerten 0 gefüllt ist, so eingefügt, dass sie sich bei Verwendung der Y-Komponente der Translation des Slices mit der kleinsten Y-Komponente der Translation (im Beispiel Slice 1) an der richtigen Position im Raum befindet. Dieses Prinzip ist in Abbildung 40 verdeutlicht.

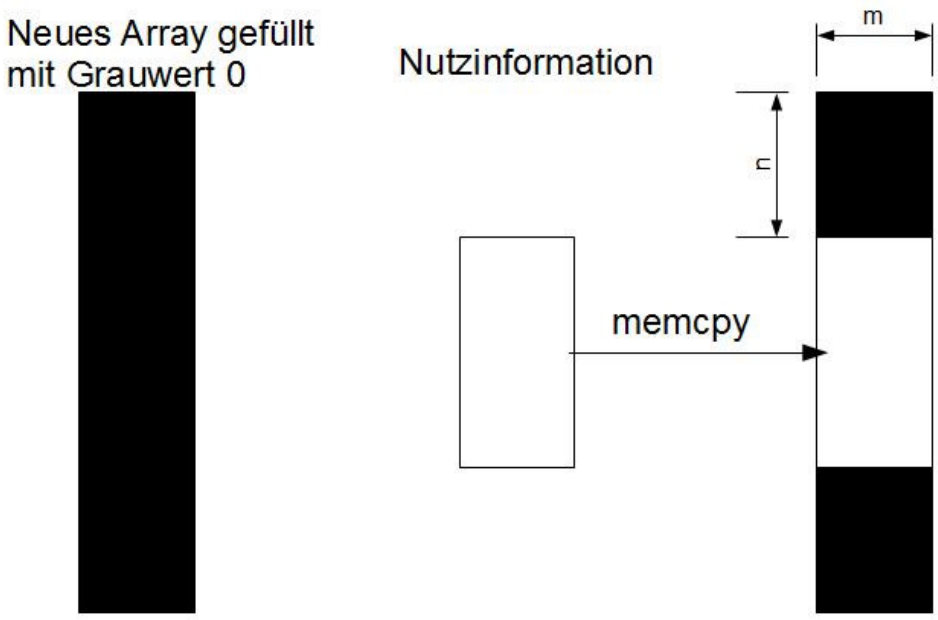

Abbildung 40: Einfügen der Nutzinformation mit memcpy

Der Algorithmus ordnet das neue Array dem vtkScalarVolumeNode zu, der das aktuell modifizierte Bild enthält. Außerdem wird die Y-Komponente der Translation des Bildes durch die des Bildes mit dem geringsten Wert der Y-Komponente der Translation ersetzt (im Beispiel Slice 1). Die Schritte werden für jedes Bild der Serie wiederholt. Am Ende haben alle Bilder dieselbe Ausdehnung in X- und Y-Richtung. Die Transformationen der einzelnen Bilder unterscheiden sich lediglich durch ihre Z-Komponente. Auch die Nutzinformation befindet sich räumlich gesehen an der richtigen Position.

Deshalb können die Bilder anschließend zu einem Volumendatensatz kombiniert werden. Der in Abbildung 39 ungenutzte Raum im Quader oberhalb und unterhalb der einzelnen Bilder ist nun bei allen Slices mit dem Grauwert 0 gefüllt, der mit keinerlei Reflexionen korrespondiert.

Um die beschriebenen Berechnungen durchführen zu können müssen zuvor weitere Schritte durchlaufen werden. Volume Builder Module kann wie in Kapitel 5.2.2. beschrieben eine Liste mit mehreren Bildserien halten. Um nur die Bilder einer Serie zu modifizieren, muss der Algorithmus diese identifizieren. Ein Bild gehört zu einer Serie, wenn die Rotationen der Transformationen der Bilder identisch sind und sich die X-Komponenten in den Bildkoordinaten der einzelnen Bilder nur um einen Toleranzwert zum Ausgleich der Roboterungenauigkeit und von Ungenauigkeiten die daraus resultieren, dass die Transponierte eine Matrix, die zum Wechseln der Basis verwendet wird, nicht numerisch stabil zu berechnen ist, unterscheiden. Der Algorithmus stellt hierzu die Anforderung an die Liste, dass alle Bilder einer Serie direkt an aufeinanderfolgenden Indizes in der Liste liegen müssen. Findet der Algorithmus ein Bild, das zu einer neuen Serie gehört, wird die alte Serie automatisch geschlossen. In der Praxis ist diese Einschränkung von keinerlei Relevanz, da jedes Bild das beim Fahren einer Trajektorie entsteht, zur gleichen Serie gehört. Die Trajektorien werden nicht unterbrochen, sondern immer vollständig abgearbeitet, so dass alle Bilder in der Liste von Volume Builder Module hintereinander geschrieben werden.

Außerdem beziehen sich die an Slicer übertragenen und den Bildern zugeordneten homogenen Koordinaten nicht auf das Koordinatensystem der Sonde, sondern auf das Basiskoordinatensystem des Roboters. Um die beschriebenen Berechnungen durchzuführen, müssen die Koordinaten jedes Bildes in das Koordinatensystem der Sonde umgerechnet werden. Hierzu wird vor jeder Verwendung einer Transformation ein Basiswechsel wie in Kapitel 2.3. beschrieben vorgenommen. Die Operationen für das Transponieren der Rotationsmatrizen und die Multiplikation von Matrizen und Vektoren werden hierbei von der Klasse vtkMatrix3x3 übernommen, die in VTK enthalten ist. Nach dem Wechseln der Basis ist die Transformation im Koordinatensystem des Bildes/der Sonde bekannt und kann wie oben beschrieben verwendet werden.

Das Koordinatensystem der Roboterbasis wird in 3D Slicer verwendet, da die von dem in Kapitel 5.3.4. beschriebenen Algorithmus generierten Volumendatensätze relativ zum Roboter in der 3D Szene von Slicer positioniert und gegeneinander registriert werden sollen. Die richtige Positionierung im Raum und Registrierung erfolgt bei Verwendung der Roboterbasiskoordinaten durch die fixe Position der Roboterbasis automatisch.

Der Quellcode des Algorithmus ist in Anhang C hinterlegt.

5.4.4. Algorithmus zur Generierung der Volumendatensätze

Der letzte Schritt der Bildverarbeitungskette im 3D Slicer Modul Volume Builder generiert aus den vorverarbeiteten 2D Ultraschallbildern Volumendatensätze. Hierzu wird mit dem in 5.3.3. beschriebenen Algorithmus die aktuelle Bild-, Transformations- und Parametersatzliste nach Bildserien durchsucht. Die Anforderungen an die Liste entsprechen denen des Algorithmus in 5.3.3. Nachdem die Bilder einer Serie gefunden wurden, werden die Dimensionen des Volumendatensatzes berechnet. Die Ausdehnung auf der X- und der Y-Achse entsprechen denen der Ausdehnung eines einzelnen vorverarbeiteten Bildes der Serie. Die Ausdehnung auf der Z-Achse ergibt sich durch die Anzahl der Bilder in der Serie und den Abständen zwischen diesen (Länge der Trajektorie).

Anschließend wird ein neuer vtkScalarVolumeNode erzeugt, dem die Transformation des ersten Bildes der Serie zugeordnet wird. Die Spacings für diesen Knoten, also die Abstände eines Pixelmittelpunktes zu dem des Nachbarpixels, werden für X- und Y-Achse von den Spacings in den 2D Ultraschallbildern übernommen. Der Abstand der Pixelmittelpunkte (Breite eines Pixels) auf der Z-Achse berechnet sich über die Distanz zwischen zwei Bildern in der Serie. Dies entspricht dem Betrag der Differenz der Z-Komponenten der Translationen zweier benachbarter Bilder. Für den neuen vtkScalarVolumeNode wird ein Array erzeugt, das die Grauwerte der einzelnen Bilder beinhaltet. Mittels memcpy werden die Bilder in den neuen Volumendatensatz kopiert. Das erste Bild wird ohne Offset in das Array kopiert. Danach wird der Offset um n*m Stellen im Array verschoben, wobei n die Anzahl der Zeilen und m die Anzahl der Spalten in einem Ultraschallbild sind, und das nächste Bild an die neue Position kopiert. Dieser Vorgang wird solange wiederholt, bis alle Bilder der Serie in das Array geschrieben sind. Abbildung 41 zeigt diesen Kopiervorgang.

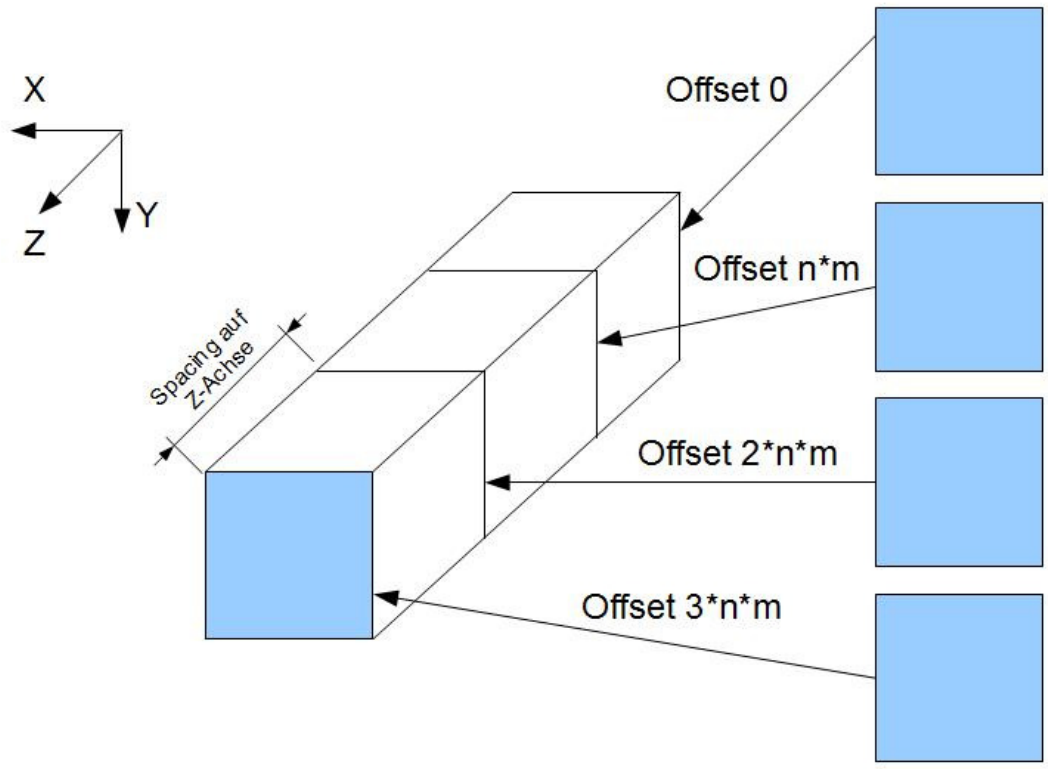

Abbildung 41: Kopieren der Slices in den Volumendatensatz

Die Spacings auf der Z-Achse sind durch äquidistante Schritte des Roboters auf der Trajektorie konstant. Deshalb reicht es aus, aus den Abständen zweier Bilder einer Serie die Spacings auf der Z-Achse auszurechnen und dem Volumendatensatz die Transformation des ersten Bildes der Serie zuzuordnen, um einen unverzerrten Volumendatensatz zu erzeugen, dessen Slices sich an der korrekten Position im kartesischen Raum befinden.

Der Algorithmus wiederholt alle Schritte, bis die komplette Liste der Bild-, Transformations- und Parametersätze ist abgearbeitet wurde, die aktuell geladen ist. Am Ende werden alle erstellten Volumendatensätze in die aktuelle MRML-Szene von Slicer übernommen und können fortan in Slicer visualisiert und bearbeitet werden. Der Aufbau eines 3D-Ultraschalldatensatzes, der mittels des Ultraschalltomographen und der vorgestellten Algorithmen erstellt wurde, unterscheidet sich hinsichtlich des Aufbaus nicht von Datensätzen, die mit anderen Modalitäten wie CT- oder MRT-Geräten erstellt wurden.

Der Quellcode des Algorithmus ist in Anhang D hinterlegt.

6. Ergebnisse

In Kapitel 6 werden die Anwendung des Ultraschalltomographen und die gewonnen Ergebnisse beschrieben. Die generierten Volumendatensätze wurden unter Verwendung von 0.1 mm - 1 mm Schichtabstand gewonnen. Es wurden einerseits Messungen an Phantomen vorgenommen, um bekannte Geometrien mittels Ultraschall abzubilden, andererseits wurden aber auch Messungen am Arm des Autors und weiterer Probanden vorgenommen, um zu evaluieren wie gut sich anatomische Strukturen im Körper des Menschen unter Verwendung des Ultraschalltomographen abbilden und beurteilen lassen. Zur Evaluierung der Genauigkeit wurde ein Testkörper mechanisch vermessen und anschließend mit dem Ultraschalltomographen gescannt. Die Ergebnisse wurden schließlich verglichen.

6.1. Messungen an Phantomen

Die Messungen an Phantomen erfolgten größtenteils unter Verwendung des RX90. Der RX90 bietet, wie in den Kapiteln 4.6. und 4.7. beschrieben, im Vergleich zum Kuka LWR IV eine bessere Wiederholungsgenauigkeit. Der RX90 kann aufgrund fehlender Kraftmomentensensorik nur verwendet werden, wenn das Phantom ohne direkten Kontakt zur Ultraschallsonde und damit ohne mechanische Verbindung zwischen Roboter und Phantom gescannt werden kann. Diese Bedingung wird von Phantomen eingehalten, die sich im Wasserbad versenken lassen. Wasser bietet hervorragende Ankopplungseigenschaften und wirkt als Vorlaufstrecke für die Ultraschallbildgebung. Das Phantom wird für die Erfassung vollständig in einem Wasserbad versenkt und die Ultraschallsonde soweit ins Wasserbad eingetaucht, dass die Ankopplungsfläche der Sonde komplett unter Wasser ist, aber kein mechanischer Kontakt zum Phantom hergestellt ist. Ist die Eindringtiefe für das Ultraschallgerät genügend groß gewählt und liegt das Phantom in Verlängerung der Ankopplungsfläche, werden aufgrund der hervorragenden Ankopplungseigenschaften von Wasser die Strukturen in den Phantomen sichtbar.

Abbildung 42 zeigt die Wirkung des Wassers als Vorlaufstrecke.

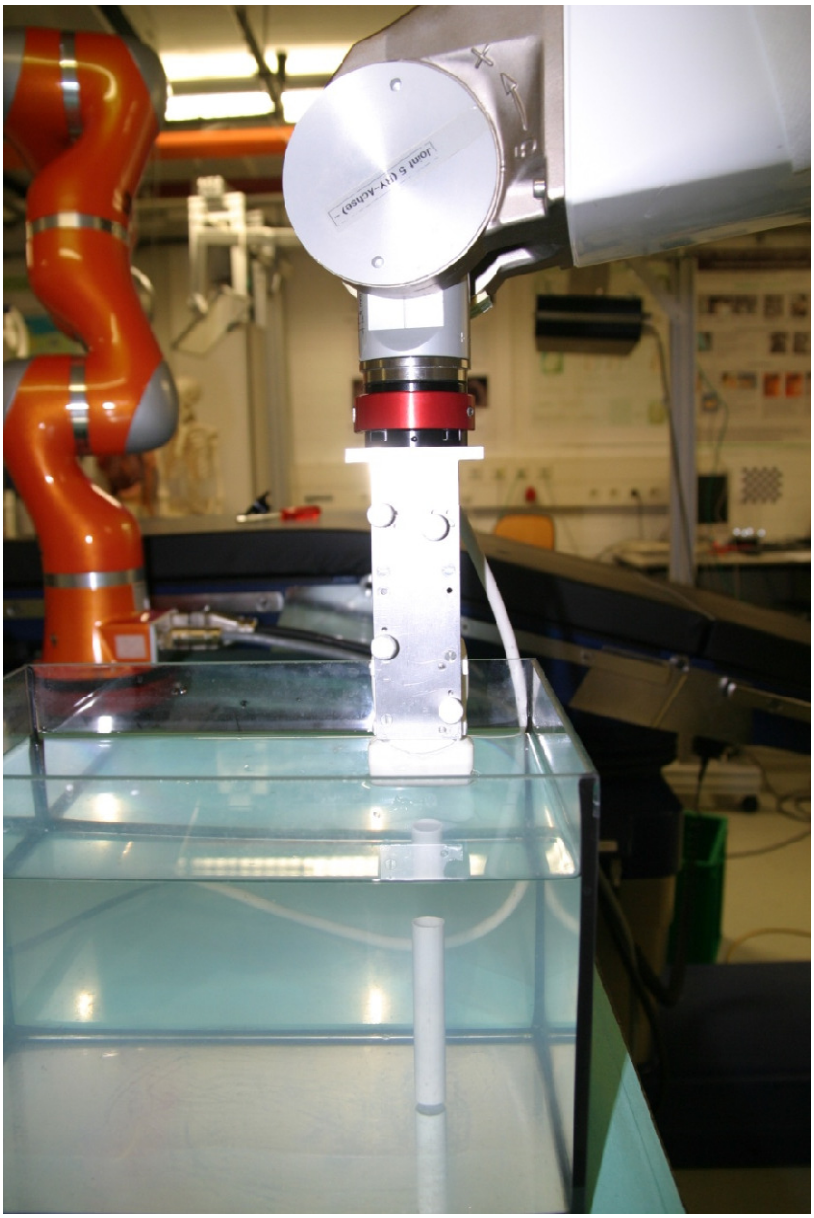

Abbildung 42: Prinzip des Scannens im Wasserbad

Durch Verwendung des Wasserbades als Vorlaufstrecke kann mit dem RX90 eine lineare Trajektorie über das Phantom gefahren werden, ohne dass das Phantom von der Sonde berührt wird. Ein weiterer Vorteil des Wasserbades ergibt sich bei der Messung an Phantomen, die mit Luft gefüllt sind, oder sich aufgrund ihrer Form nur unzureichend akustisch an die Sonde ankoppeln lassen (schlechter Kontakt zur Ankopplungsfläche der Sonde). Das Wasser verdrängt hierbei die Luft aus den Hohlräumen und sorgt für eine bestmögliche akustische Ankopplung.

Um bessere Ergebnisse zu erhalten, wurde ausschließlich destilliertes Wasser verwendet. Dieses ist blasenfrei und verursacht somit weniger Reflexionen als Leitungswasser. Die Schallgeschwindigkeit in einer Flüssigkeit ist abhängig von deren Kompressionsmodul, das wiederrum abhängig von der Temperatur des Mediums ist. Aufgrund der in Ultraschallgeräten standardmäßig verwendeten Fortleitungsgeschwindigkeit von 1540 m/s ist, um Verzerrungen zu vermeiden, das Wasser auf eine Temperatur zu bringen, bei der die Fortleitungsgeschwindigkeit möglichst nahe bei den

vorgegebenen 1540 m/s liegt. Dies ist beim circa 49 Grad Celsius[47] der Fall. Während dieser Arbeit wurde eine Heizung für Aquarien verwendet, mit deren Hilfe eine Wassertemperatur von circa 34 Grad Celsius erreicht werden konnte. Die Fortleitungsgeschwindigkeit in Wasser liegt bei 34 Grad Celsius bei etwa 1520 m/s. Die geringe Abweichung vom 20 m/s wurde in Kauf genommen. Mittels des in den Kapiteln 4 und 5 vorgestellten Ultraschalltomographen wurden Volumendatensätze vielfältiger Phantome erstellt. In einem anderen am Institut durchgeführten Projekt wurde ein ultraschallfähiges Phantom aus Silikon hergestellt. Dieses in Abbildung 43 dargestellte Phantom ist das Modell einer menschlichen Leber, das von zahlreichen Kunststoffkapillaren durchzogen ist.

Abbildung 43: Leberphantom aus Silikon

Da Silikon schwimmfähig ist, wurde das Phantom durch einer Aluminiumplatte unter Wasser beschwert, um die Position am Beckenboden konstant zu halten. Der Versuchsaufbau ist in Abbildung 44 zu sehen.

[47] http://www.lsbu.ac.uk/water/explan2.html#sound (letzter Zugriff: 01.12.2010)

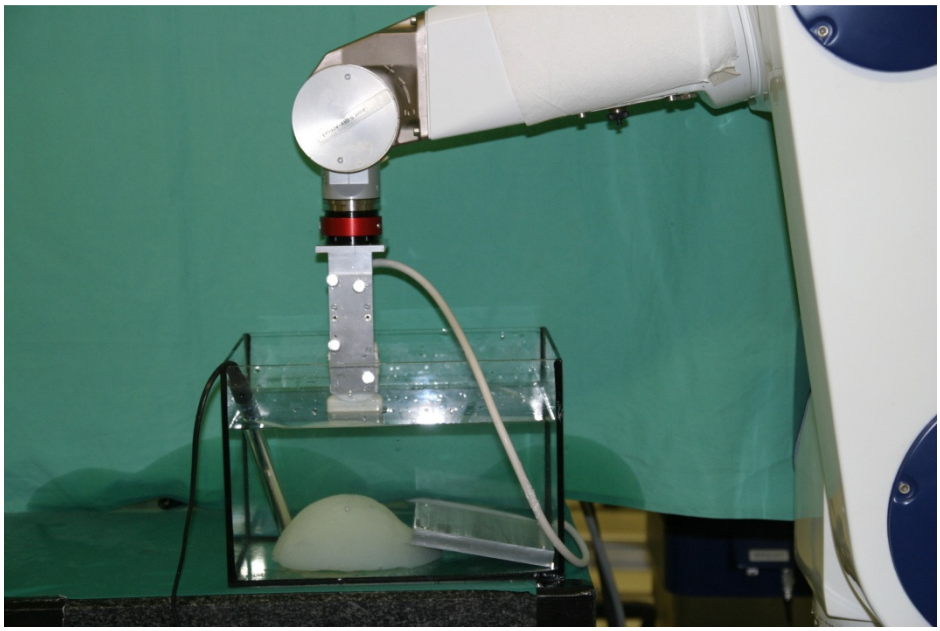

Abbildung 44: Versuchsaufbau mit Leberphantom

Mittels einer linearen Trajektorie im Wasserbad wurde unter Verwendung des Ultraschalltomographen ein Volumendatensatz des Leberphantoms erzeugt. Dieser Datensatz wurde unter Verwendung eines Schichtabstandes von 1 mm erstellt und zeigt sowohl die Oberfläche des Leberphantoms, als auch eine durchgängige Kapillare innerhalb des Phantoms. Der Datensatz wurde unter Verwendung von Volume Rendering mit einer Compositing Methode in 3D visualisiert.

Abbildung 45 zeigt die dreidimensionale Visualisierung zusammen mit drei repräsentativen Slices in den Standardorientierungen axial (links), sagittal (Mitte) und coronal (rechts), wobei die zweidimensionalen Einzelbilder der Ultraschallsonde aller in diesem Kapitel behandelten Datensätze in der coronalen Ebene liegen und somit einzeln in 3D Slicer begutachtet werden können. In der Abbildung wird das 29. Einzelbild der Sonde dargestellt. Die multiplanare Rekonstruktion zeigt in allen drei Ansichten die Oberfläche des Phantoms sowie eine der Kunststoffkapillaren. Für das Leberphantom existiert weder ein Referenzdatensatz aus einem CT- oder MRT, noch ein CAD-Modell mit genauen Abmessungen, so dass dieser Datensatz nur qualitativ begutachtet werden kann.

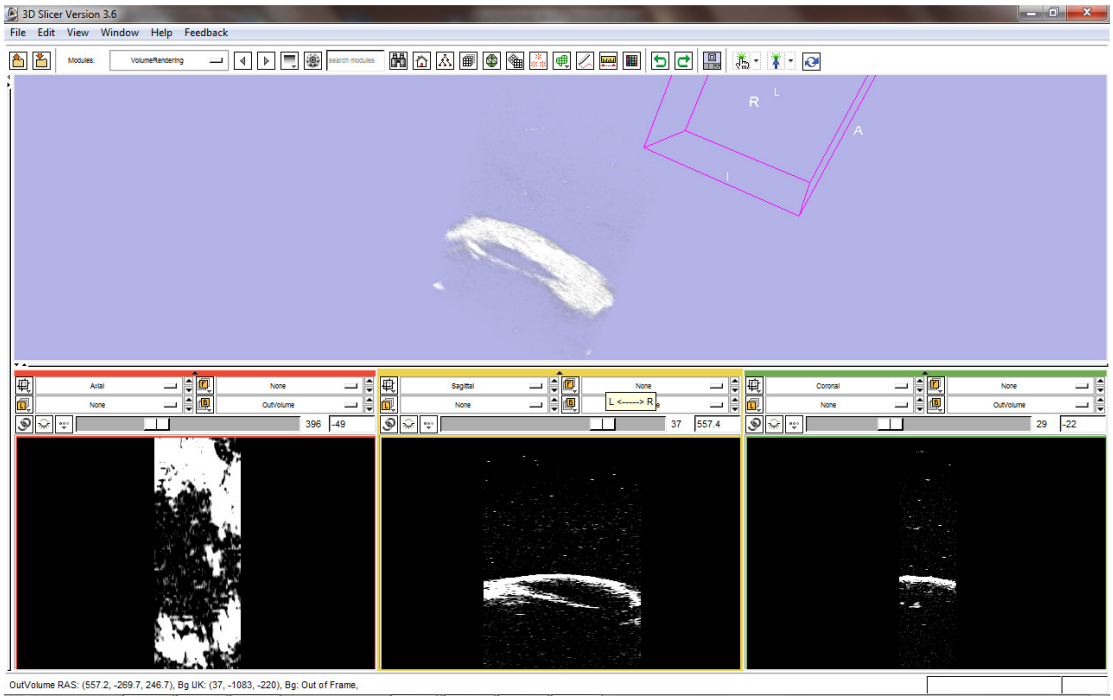

Abbildung 45: Visualisierung des Leberphantoms in 3D Slicer

Um zu evaluieren ob der Ultraschalltomograph korrekte Volumendatensätze liefert, wurde ein zylindrisches Objekt als weiteres Testobjekt gesucht. Als Testobjekt wurde schließlich ein Schlagstock eines Schlagzeuges gewählt. Dieser enthielt mehrere Testbohrungen. Holz eignet sich eher schlecht für die Bildgebung mittels medizinischer Ultraschallmodalitäten. Deshalb konnte nur die der Ultraschallsonde in axialer Richtung näher liegende Seite des Schlagstabes visualisiert werden. Die Zylinderform des Objektes ist deutlich zu erkennen. Der Datensatz wurde korrekt akquiriert und berechnet. Die verwendete Auflösung in Richtung der Trajektorie beträgt 0.1 mm. Abbildung 46 zeigt den Volumendatensatz des Schlagstockes mit erkennbarer Oberfläche und Bohrung.

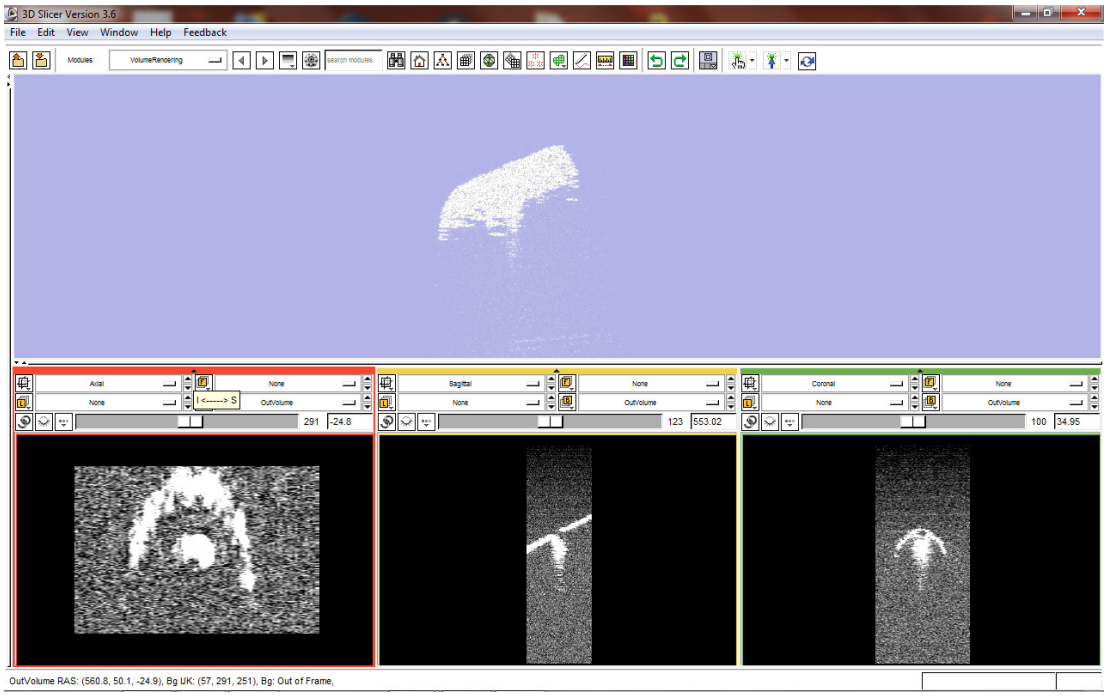

Abbildung 46: Visualisierung des Schlagstockes

Andere Phantome konnten aufgrund ihrer Materialeigenschaften nicht im Wasserbad versenkt werden, so zum Beispiel das am Institut vorhandene Multimodalphantom, das zur Bildgebung mittels CT und Ultraschall geeignet ist. Das Phantom stellt ein menschliches Abdomen dar und ist aus Kunststoff gefertigt. Enthalten sind Modelle von Wirbelsäule, Rippen, Niere und Leber. Das Phantom ist in Abbildung 47 zu sehen.

Abbildung 47: Multimodalphantom des Abdomen

Um ein solches Phantom zu scannen, muss eine mechanische Verbindung zwischen Phantom und Ultraschallsonde hergestellt werden. Dies kann mittels einer kraftbasierten Trajektorie realisiert werden, wie sie in Kapitel 5 beschrieben ist. Der Schallkopf folgt hierbei robotergeführt der Oberfläche eines Objektes. Die akustische Ankopplung wird wie bei der Ankopplung zwischen Sonde und Haut durch Ultraschallgel auf Wasserbasis realisiert. Mittels des Ultraschalltomographen wurde eine der Rippen innerhalb des Phantoms visualisiert. Die Auflösung in Richtung der Trajektorie beträgt 1 mm.

Das Visualisierungsergebnis ist in Abbildung 48 zu sehen.

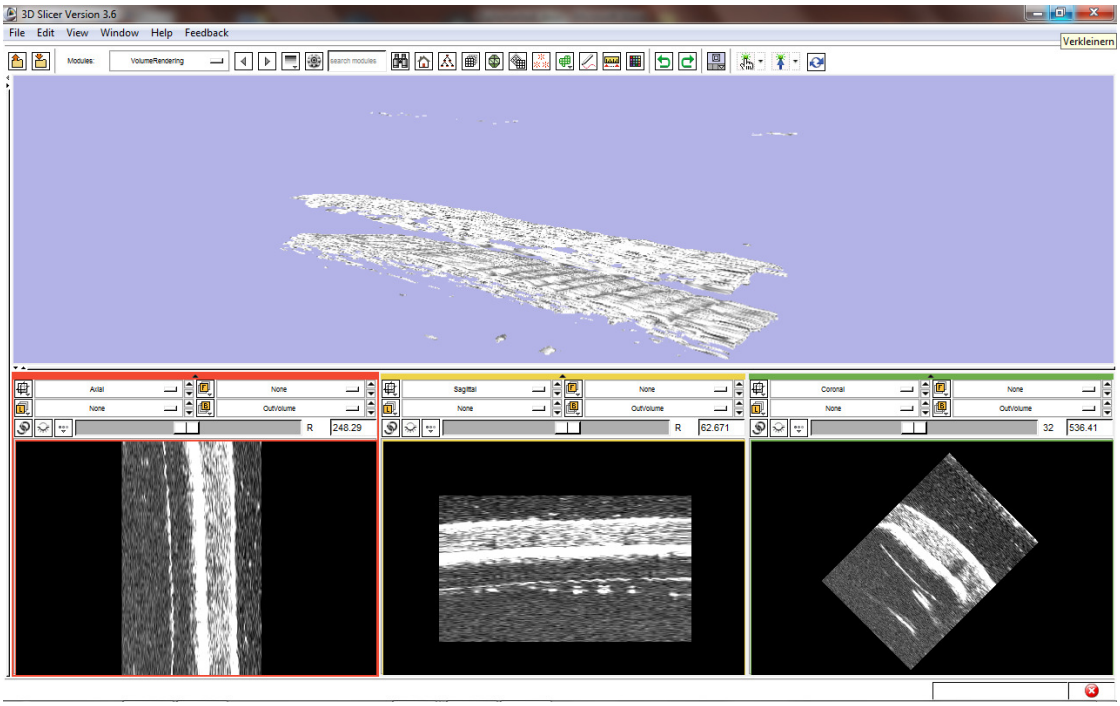

Abbildung 48: Visualisierung einer Rippe innerhalb eines Multimodalphantoms

6.2. Evaluierung der Genauigkeit

Zur Evaluierung der Genauigkeit wurden Phantome mit bekannter Geometrie oder Phantome zu denen Referenzdatensätze existieren verwendet. Im konkreten Fall wurde ein Kunststoffmodell eines menschlichen Schädels verwendet, von dem ein CT-Datensatz existiert. Außerdem wurde ein Kunststoffrohr mittels eines Messschiebers mit 1/100 mm Skala vermessen und mit den Messungen innerhalb verschiedener Volumendatensätze verglichen, die mit dem Ultraschalltomographen erstellt wurden.

Zum Scannen des Kunststoffschädels wurde das Wasserbad verwendet. Die Schädeldecke wurde hierzu abgenommen und der Schädel anschließend ohne Unterkiefer mit dem Oberkiefer und Os occipitale nach oben in das Wasserbad gelegt. Das Scannen erfolgte unter Verwendung einer Auflösung von 1 mm in Richtung der Trajektorie. Der Versuchsaufbau ist in Abbildung 49 zu sehen. Mit der verwendeten Trajektorie konnte ein Volumendatensatz erstellt werden, der die Unterseite des Schädels von den Schneidezähnen bis zum Hinterhaupt abdeckt.

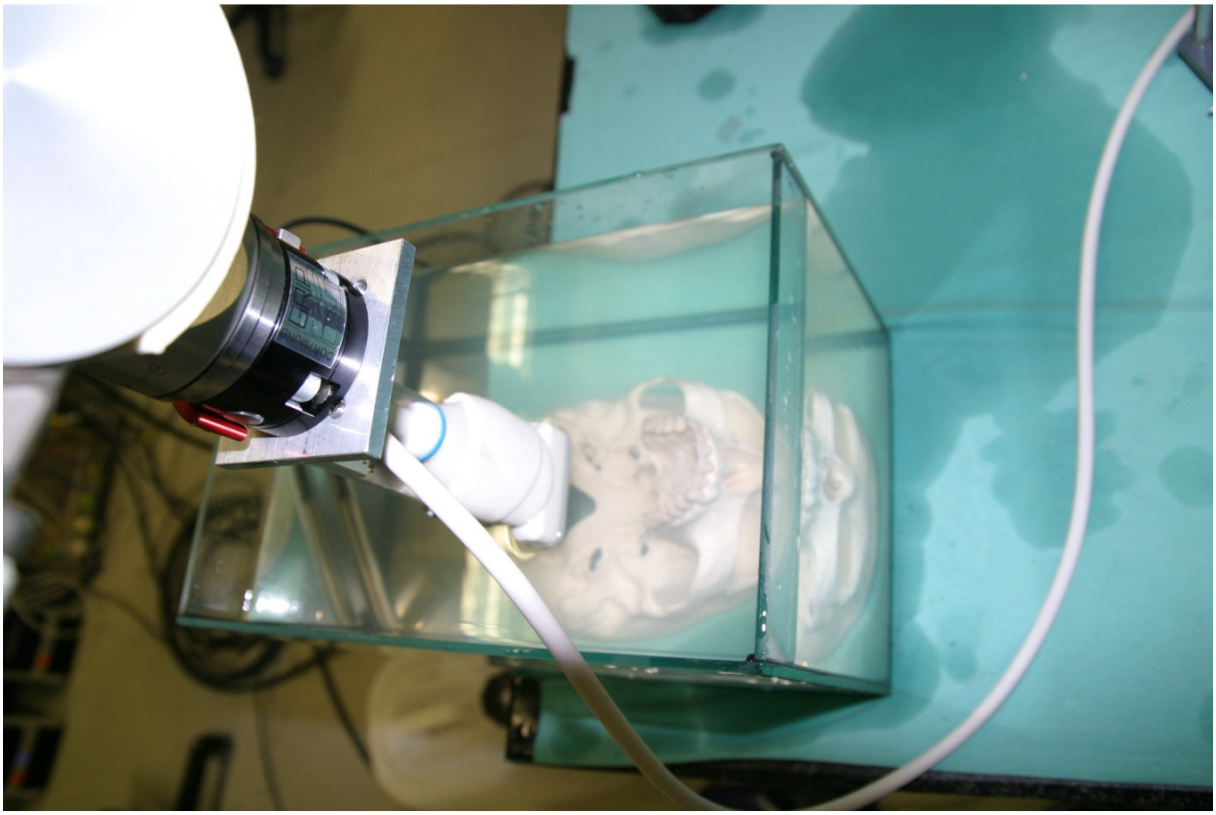

Abbildung 49: Scannen des Schädels

Um die Genauigkeit des Ultraschalltomographen zu evaluieren, wurde zusätzlich zum Volumendatensatz aus dem Ultraschalltomographen ein CT-Datensatz des Schädels in die Slicer Szene geladen. Dem CT-Datensatz wurde ein Transformationsknoten zugeordnet und die entsprechende Transformation so lange von Hand verändert, bis ein guter Überlapp zwischen beiden Datensätzen festzustellen war. Hierzu bietet Slicer die Möglichkeit, multiplanare Rekonstruktionen verschiedener Volumen gleichzeitig zu berechnen und mit verschiedener Transparenz in derselben Ansicht darzustellen. Abbildung 50 zeigt das Registrierungsergebnis zwischen CT- und

Ultraschalldatensatz. Die Transparenz der beiden Datensätze liegt in allen Ansichten bei circa 50%. Es werden die multiplanaren Rekonstruktionen und ein volumengerendertes Bild des CT-Datensatzes in der 3D-Ansicht von Slicer dargestellt.

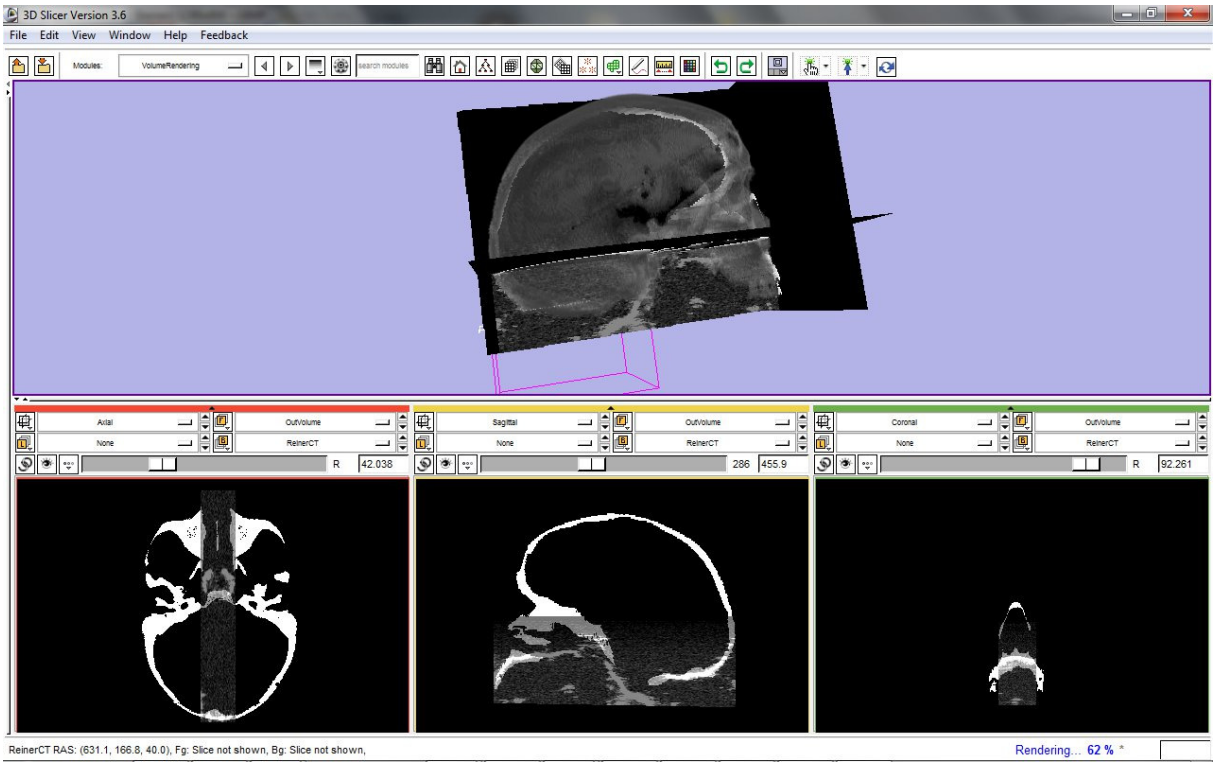

Abbildung 50: Registrierungsergebnis zwischen Ultraschall- und CT-Datensatz

Der Überlapp zwischen Ultraschall- und CT-Datensatz ist besonders gut in der Sagittalebene zu sehen. Ein Slice in der Sagittalebene ist in Abbildung 51 vergrößert dargestellt.

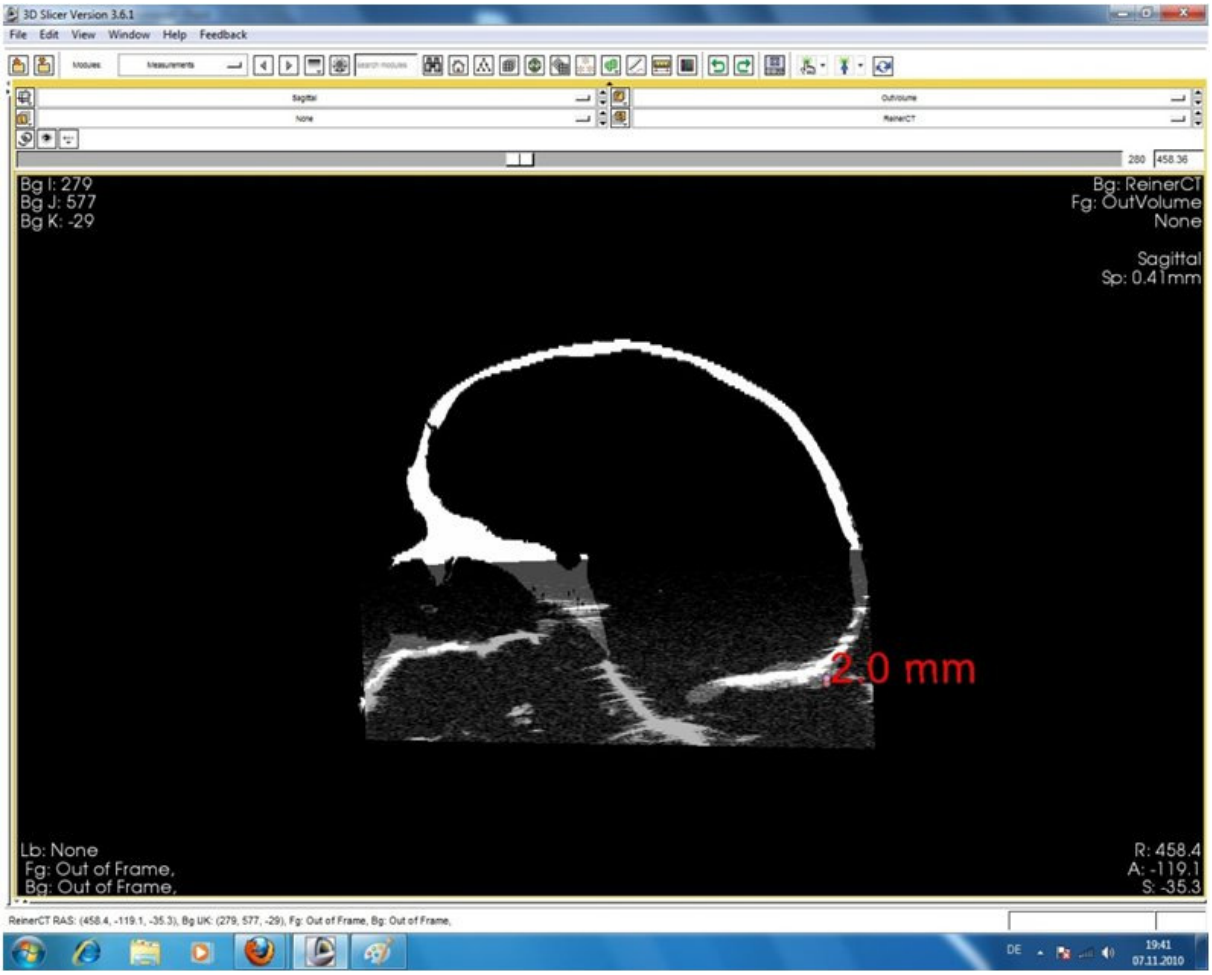

Abbildung 51: Vergrößerte Slices in der Sagittalebene

Die betrachteten Slices von Ultraschall- und CT-Datensatz in Abbildung 51 zeigen in den meisten Bereichen einen guten Überlapp zwischen den Datensätzen. Dennoch gibt es vor allem im Bereich des Os occipitale Abweichungen in der Größenordnung von mehreren Millimetern. Diese Abweichungen lassen sich über die komplette Serie beobachten. Es ist an dieser Stelle nicht klar, ob die Abweichungen durch einen Registrierungsfehler oder Verzerrungen im Datensatz entstanden sind. Deshalb wurde ein Kunststoffrohr mit 15.3 mm Außendurchmesser, 13.5 mm Innendurchmesser und 1 mm Materialstärke mechanisch mittels eines Messschiebers und akustisch mittels Ultraschall vermessen. Es wurden zwei Datensätze erstellt, bei denen das Rohr in Längs- und Querrichtung gescannt wurde, um so die Genauigkeit in allen Richtungen evaluieren zu können.

Während ein einzelnes Ultraschallbild eines im Wasserbad liegenden Rohres den Querschnitt des Rohres zeigt, zeigt ein einzelnes Bild des im Wasserbad stehenden Rohres den Längsschnitt des Rohres. Das Vermessen des Rohres im Ultraschallbild war nicht mit ausreichender Genauigkeit möglich. Die Materialstärke des Rohres schwankte in einem einzelnen Ultraschallbild zwischen 2.2 und 3 mm. Die Materialstärke ist auch in der Realität nicht konstant. Beim Vermessen mittels eines Messschiebers konnten Materialstärken zwischen 0.82 und 1 mm festgestellt werden. Durch die Materialeigenschaften und die begrenzte Auflösung des Ultraschallgerätes wurde die Materialstärke in den Bildern trotz optimaler Fokussierung verfälscht dargestellt. Um dennoch Rückschlüsse auf die Genauigkeit des Tomographen zu ermöglichen wurden beide Volumen gemeinsam in 3D Slicer

visualisiert. Die Volumen wurden gegeneinander um 90 Grad verdreht und so verschoben, dass sich die Querschnitte des Rohres in den Datensätzen bestmöglich überlappen.

Das Prinzip ist in Abbildung 52 verdeutlicht

.

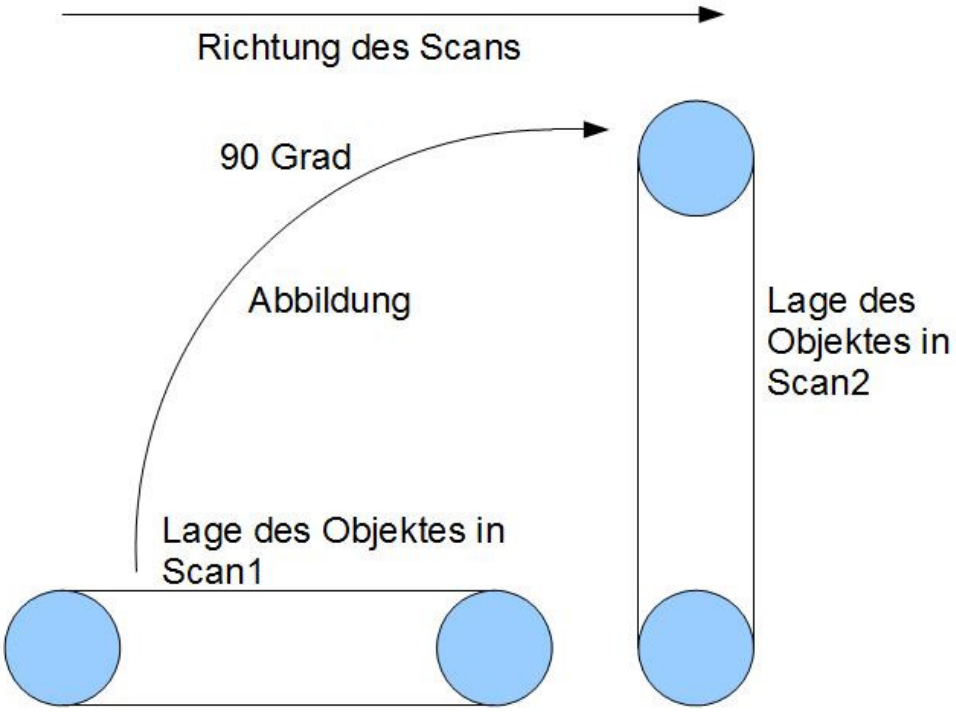

Abbildung 52: Abbildung der Querschnitte desselben Objektes bei verschiedenen Lagen des Objektes aufeinander

Die Datensätze wurden dadurch gegeneinander registriert. Das Registrierungsergebnis ist in Abbildung 53 zu sehen. Die Slices zweier Datensätze werden hier wie auch bei Abbildung 51 gemeinsam mit circa 50% Transparenz dargestellt. Während bei Betrachtung des Querschnitts im Datensatz, der bei liegendem Rohr im Wasserbad erstellt wurde, das axiale und laterale Auflösungsvermögen der Ultraschallsonde beurteilt werden kann, kann bei Betrachtung des Querschnitts im Datensatz, der bei stehendem Rohr im Wasserbad erstellt wurde, das laterale Auflösungsvermögen der Sonde und die Genauigkeit der Trajektorie und damit die Genauigkeit des 3D-Datensatzes in Richtung der Z-Achse beurteilt werden.

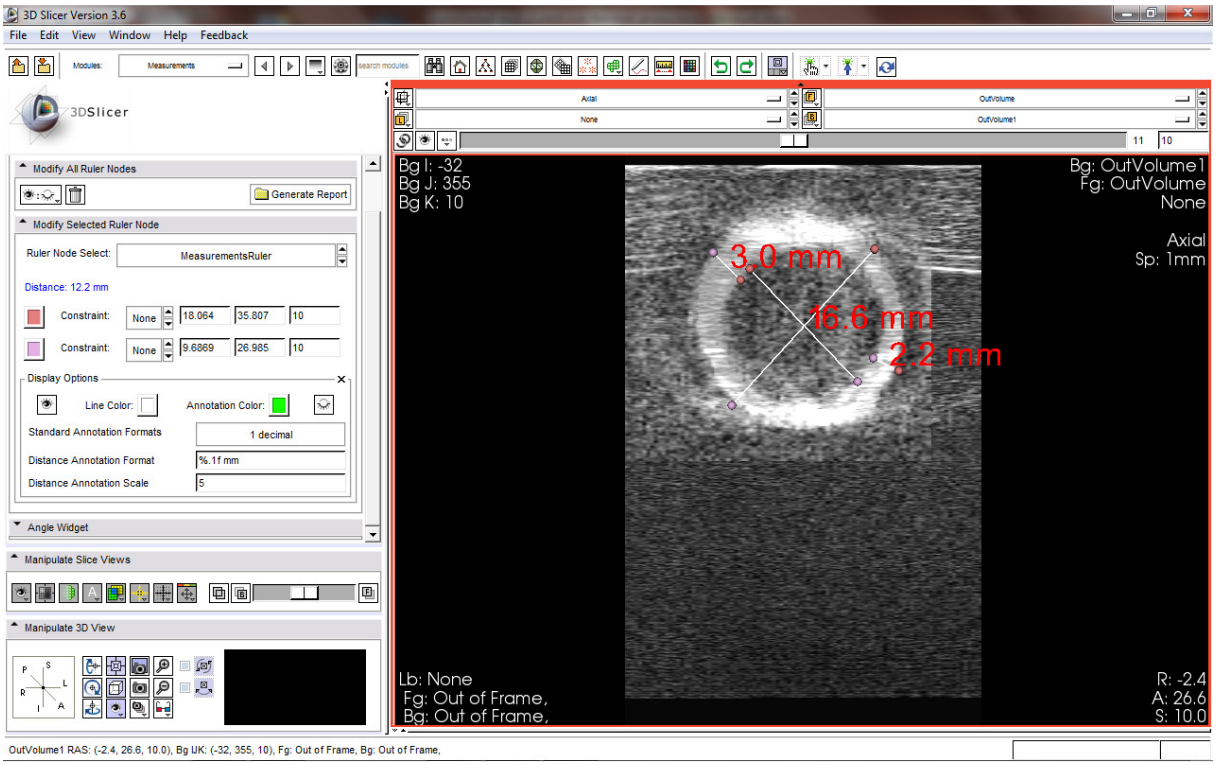

Abbildung 53: Registrierung zweier Scans mit unterschiedlicher Objektorientierung

Wie in Abbildung 53 zu sehen, überlappen sich die Querschnitte an allen Positionen in den entsprechenden Slices. Es ist keinerlei Unterschied hinsichtlich der Genauigkeit zwischen dem algorithmisch rekonstruierten Querschnitt und dem 2D-Ultraschallschnittbild der Sonde festzustellen. Die Fokussierung wurde in beiden Scans so gewählt, dass die Genauigkeit in dem Bereich des Bildes am größten ist, der den Querschnitt des Rohres zeigt. Für die Genauigkeit der Ultraschallsonde wird allgemein ein Wert zwischen 4-5*λ angenommen. Bei der hier verwendeten 7.5 MHz Ultraschallsonde ergibt sich bei Verwendung von $c = \lambda * f$ und c= 1540 m/s eine Wellenlänge von 0.2 mm. Es muss von einer lateralen Auflösung im Bereich von 1-1.2 mm (5-6* λ) in der Fokuszone ausgegangen werden, da die laterale Auflösung mit zunehmender Eindringtiefe abnimmt. Dem guten Überlapp der Querschnitte aus beiden Datensätzen kann damit entnommen werden, dass die Auflösung des Ultraschalltomographen in Richtung der Z-Achse der Sonde mindestens der lateralen Auflösung der Sonde entspricht. Die Datensätze wurden in interessanten Bereichen in Richtung der X- und Y-Achse also in axialer und lateraler Richtung der Sonde nicht modifiziert. Die Gesamtgenauigkeit des Ultraschalltomographen kann der folgenden Tabelle entnommen werden.

Richtung in Sondenkoordinaten	Auflösung	Typische Werte für 7.5 MHz
X	Laterale Auflösung der Sonde	0.4 mm
Y	Axiale Auflösung der Sonde	1.2 mm
Z	Mindestens laterale Auflösung der Sonde	Mindestens 1.2 mm

Tabelle 3: Auflösungsvermögen des Ultraschalltomographen

Mit der vorhandenen Ultraschallsonde können keine präziseren Aussagen über die Genauigkeit auf der Z-Achse gemacht werden. Der Wert ist abhängig von der Strahlstärke der Ultraschallsonde und der Wiederholgenauigkeit des Roboters. Die Vermutung liegt nahe, dass die Auflösung deutlich besser ist, als die ermittelten 1.2 mm.

Durch die große Genauigkeit der Roboter treten geringere Winkelfehler auf, als dies bei weniger genauen Verfahren wie optisch verfolgtem oder Speckle basiertem 3D-Ultraschall der Fall ist. Dies ist insofern von Bedeutung, als geringe Winkelfehler an der Sonde bei großer Eindringtiefe dennoch große Abweichungen im kartesischen Koordinatensystem verursachen können. Die Genauigkeit im Vergleich zu Verfahren, die mit externen Trackingsystemen, Speckle-Dekorellation oder 2D-Arraysonden arbeiten, konnte nicht evaluiert werden, da kein solches 3D-Ultraschallsystem zur Verfügung stand.

6.3. Aufnahmen aus unterschiedlichen Winkeln

Wie in Kapitel 5 erwähnt ist es möglich, die Ultraschallsonde um die erste Fokussierung zu drehen. Mittels dieser Funktion können anatomische Strukturen aus verschiedenen Winkeln begutachtet werden. Abbildung 54 zeigt Slices aus drei Volumendatensätzen. Diese Volumendatensätze wurden mit Hilfe dreier linearer Trajektorien erstellt und von Volume Builder Module berechnet. Sie zeigen das Jochbein (Jochbogen) des Kunststoffschädels aus den Winkeln 150, 180 und 210 Grad. Der Datensatz, der im Winkel von 180 Grad erstellt wurde, wird zusätzlich mittels Volumenrendering in 3D visualisiert.

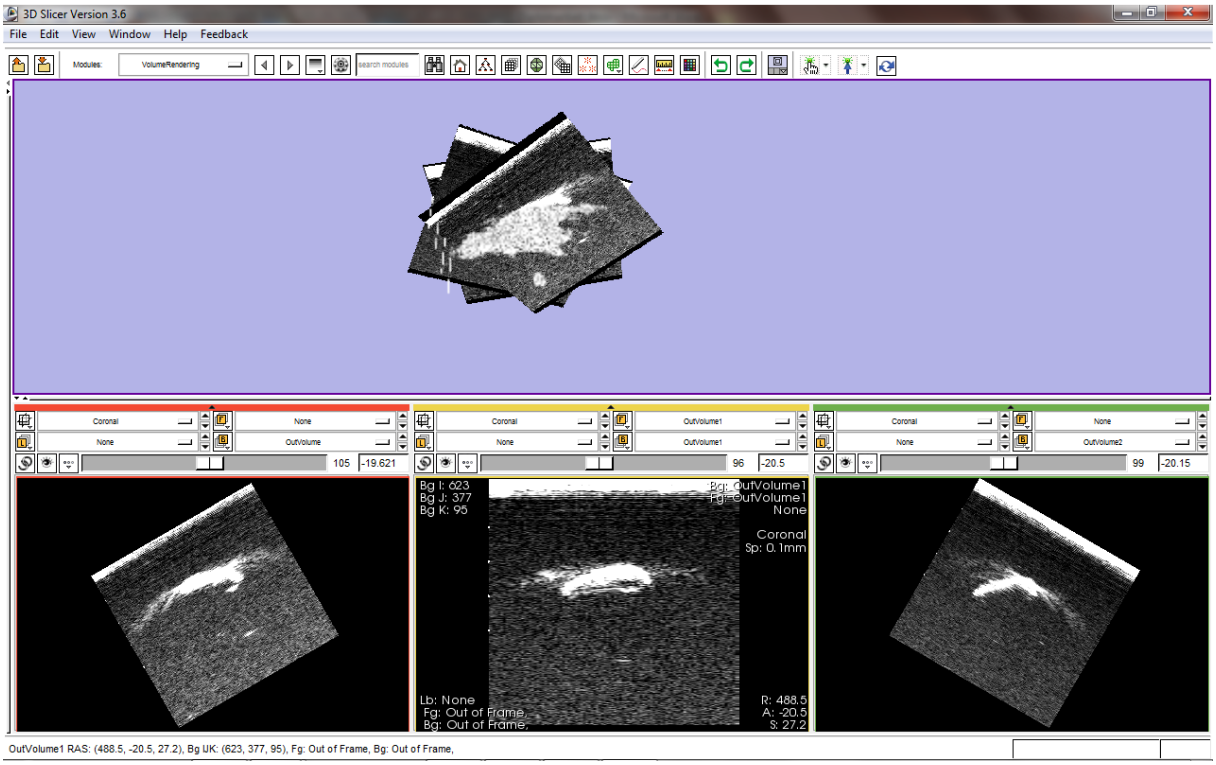

Abbildung 54: Visualisierung automatisch registrierter Datensätze

Die Datensätze konnten durch die bekannte Geometrie von Sonde und Roboter, sowie bekannte Parameter des Ultraschallgerätes automatisch gegeneinander registriert werden. In der Praxis können auf diese Weise anatomische Strukturen aus verschiedenen Richtungen begutachtet werden. Enthält ein Datensatz an einer interessanten Stelle Schallschatten oder Verzerrungen, stehen dem Anwender andere Ansichten zur Verfügung, in denen eventuell keine Schatten an der interessanten Stelle vorhanden sind. Dadurch kann ein verbessertes Untersuchungsergebnis erreicht werden. Es ist außerdem theoretisch möglich in Nachverarbeitungsschritten die einzelnen Datensätze zu einem kombinierten Datensatz zu verrechnen und damit die Bildqualität zu verbessern.

6.4. Messungen an Gewebe

Mittels des KUKA LWR IV und kraftbasierter Trajektorien wurden Volumendatensätze des menschlichen Armes gesammelt. Entstanden sind ein Datensatz der Oberseite (bei liegendem Arm und Pronation) des Unterarmes (Handrücken zeigt zur Sonde) des Autors und ein Datensatz der Unterseite (bei liegendem Arm und Supination) des Unterarmes (Handfläche zeigt zur Sonde) eines weiteren Probanden. Der Datensatz der Unterseite zeigt die Oberfläche von Radius, Muskeln, Sehnen und Gefäße. Mittels 3D Slicer wurde eine multiplanare Rekonstruktion erstellt. Einzelne Slices dieser Rekonstruktion sind in Abbildung 55 zu sehen.

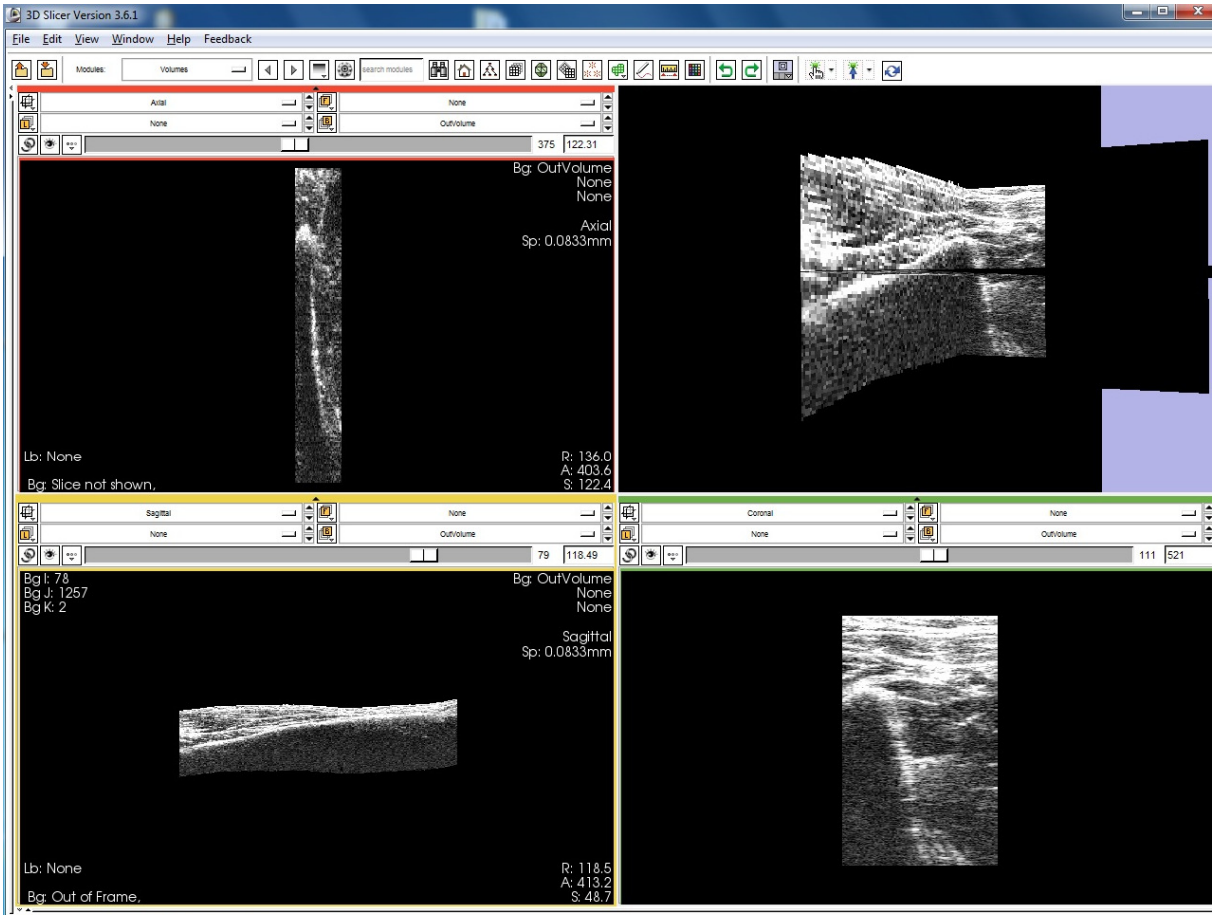

Abbildung 55: multiplanare Rekonstruktion der Unterseite eines menschlichen Armes

In allen Ansichten ist die Oberfläche des Radius zu sehen. Die coronale Ansicht zeigt den Querschnitt des Knochens und mehrerer Muskeln. Die axiale Ansicht zeigt den Längsschnitt des. Am oberen Rand der Ansicht ist zu sehen wie der Knochen in das Handgelenk übergeht. Die sagittale Ansicht zeigt ebenfalls einen Längsschnitt durch den Arm. Die am tiefsten liegende bildgebende Struktur ist die Oberfläche des Radius. Darunterliegende Strukturen sind abgeschattet und werden nicht dargestellt.

Bessere Ergebnisse konnten beim Scannen der Oberseite des Unterarmes erzielt werden. Hier konnten Radius und Muskeln längs des Armes gut sichtbar und volumengerendert in 3D dargestellt werden. Multiplanare Rekonstruktion und Volumenrendering der Oberseite des Unterarmes zeigt Abbildung 56.

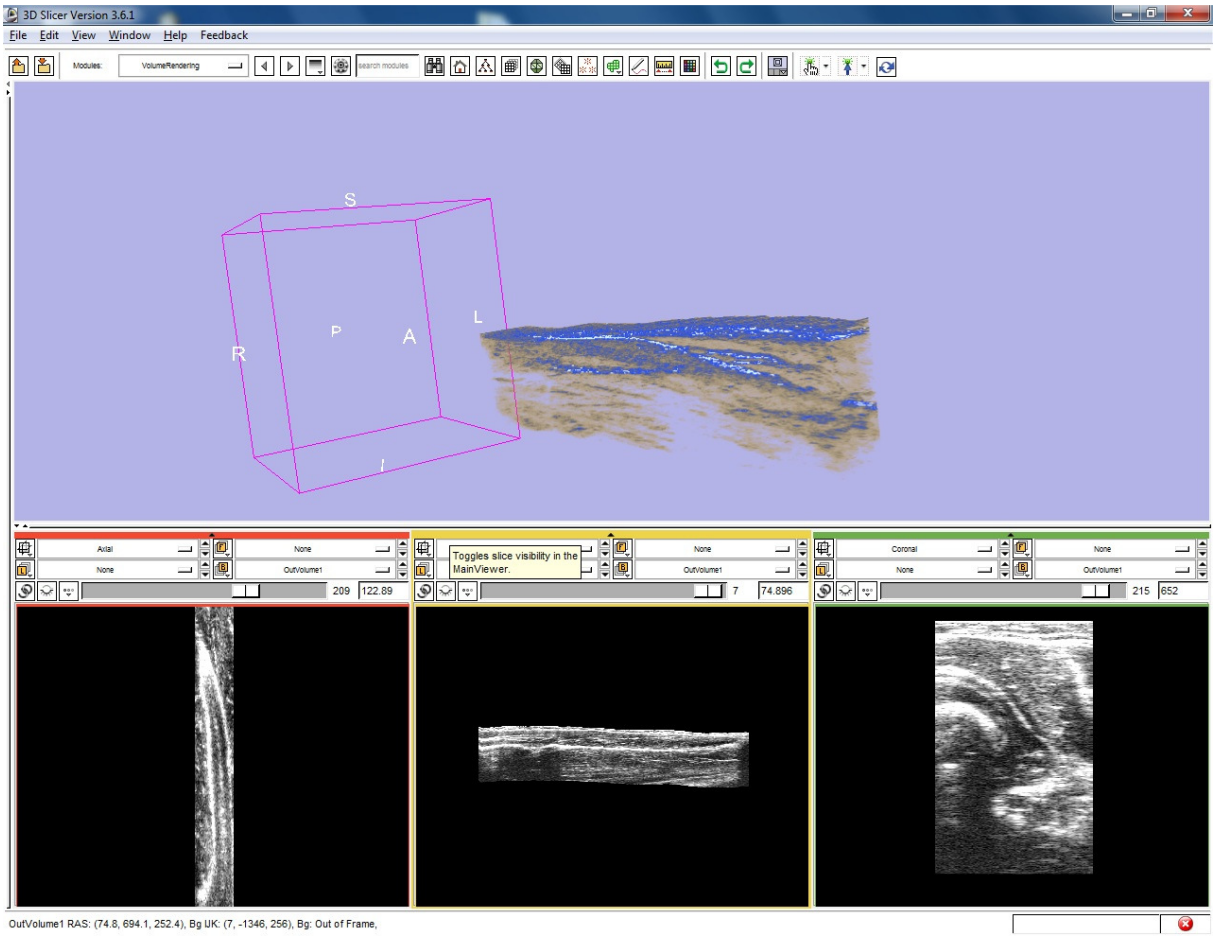

Abbildung 56: Volumendatensatz der Oberseite des Unterarmes

Axiale und sagittale Ansicht des Armes zeigen Längsschnitte des Armes. Die Kontur des Radius ist hierbei gut sichtbar. In der coronalen Ansicht zeigen sich zusätzlich Weichgewebeanteile.

Das Volumenrendering zeigt in blau die Kontur des Knochens und Teile der Ankopplungsartefakte, die nicht vollständig entfernt werden konnten. Außerdem sind ebenfalls in blau, Teile der Muskulatur visualisiert.

An Volumendatensätzen des Unterarmes erfolgten keine Evaluierungen bezüglich der Genauigkeit. Sie sollen lediglich die Verwendbarkeit des Ultraschalltomographen an Weichgewebe demonstrieren. Die Auflösung längs des Armes beträgt in beiden Datensätzen 1 mm. Der Anpressdruck des Roboters wurde so eingestellt, dass dies für die Probanden gerade noch angenehm war und betrug jeweils circa 1 kg. Der Versuchsaufbau ist in Abbildung 57 gezeigt.

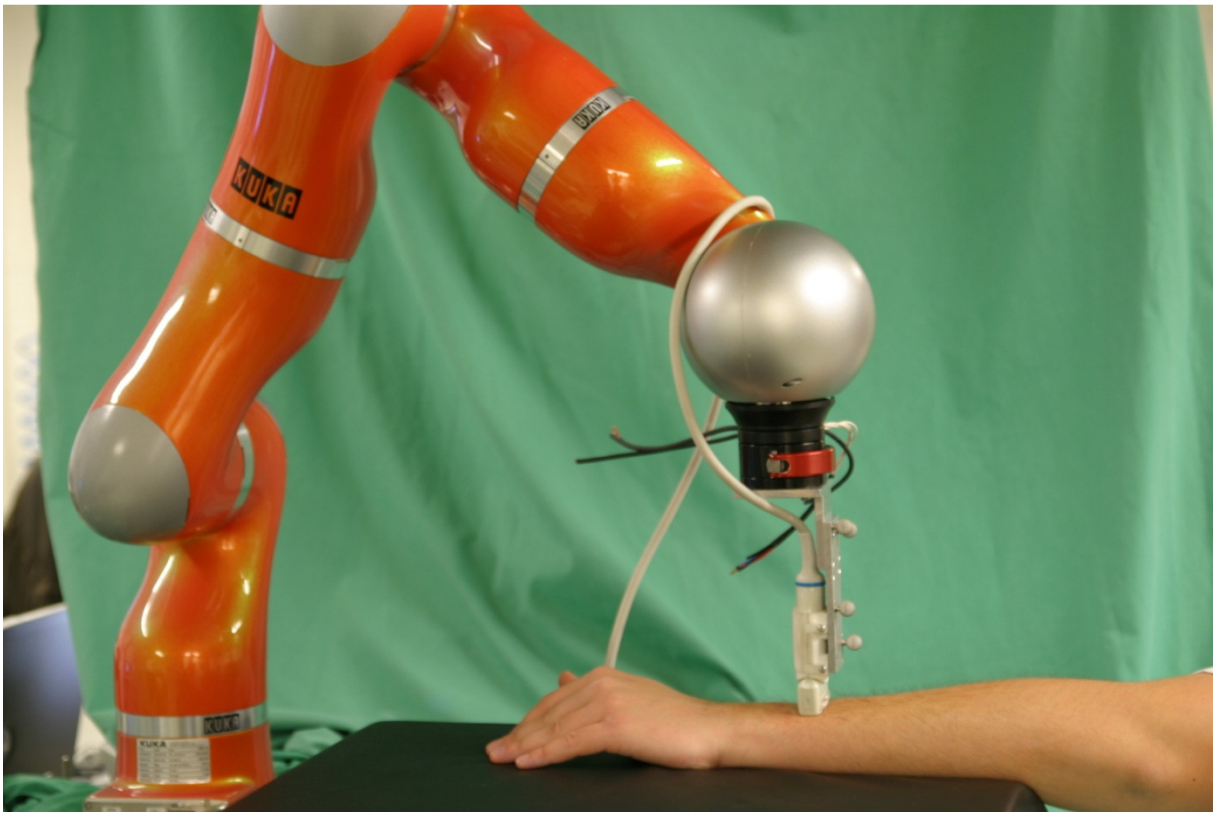

Abbildung 57: Versuchsaufbau zum Erstellen eines Volumendatensatzes an Weichgewebe eines Armes

Da der Ultraschalltomograph nicht in der Lage ist selbstständig Ultraschallgel zu applizieren, musste der komplette zu untersuchende Bereich vor dem Scan großzügig mit Ultraschallgel bedeckt werden, damit die Ankopplung zu jedem Zeitpunkt des Scanvorgangs optimal war.

7. Diskussion und Ausblick

Ziel dieser Arbeit war es einen robotergestützten Ultraschalltomographen als Entwicklungsplattform zu entwickeln. Im Folgenden werden Ergebnisse und Gesamtkonzept sowie Realisierung des Ultraschalltomographen diskutiert.

Bei der Durchführung dieser Arbeit zeigte sich, dass die Entwicklung eines Ultraschalltomographen auf Basis mehrerer Module möglich ist. Die Software des Ultraschalltomographen konnte soweit entwickelt werden, dass hochpräzise 3D-Ansichten von Phantomen und menschlichem Gewebe erstellt werden können.

Aufgrund des verwendeten Konzepts können einzelne Module des Tomographen ausgetauscht oder weiterentwickelt werden. OpenIGTLink als Kommunikationsprotokoll zwischen den einzelnen Komponenten erwies sich als als robustes Mittel zur losen Kopplung der Komponenten. Die hervorragende Integration in 3D Slicer über das OpenIGTLinkIF Modul trug zum schnellen Projektfortschritt maßgeblich bei. 3D Slicer erwies sich aufgrund des leichtgewichtigen Kerns und der vielfältigen verfügbaren Module als unkomplizierte Entwicklungsumgebung für Projekte im Kontext der medizinischen Bildverarbeitung.

Obwohl ein Großteil der Arbeit sich mit der Kommunikation zwischen den einzelnen Systemkomponenten beschäftigte, konnten die verwendeten Nachrichten einfach gehalten werden. Der Grund hierfür ist ebenfalls das schlanke OpenIGTLink Protokoll, das sich nach Belieben erweitern und in vorhandene C/C++ Projekte integrieren ließ.

Auch die Microsoft Entwicklungstools für die .Net Umgebung erwiesen sich als hervorragend geeignet für dieses Projekt. Durch die Spracherweiterungen von C++/CLI war es vergleichsweise einfach möglich, einen Wrapper für OpenIGTLink zu programmieren. Wünschenswert wäre an dieser Stelle allerdings eine bessere Dokumentation von C++/CLI, die die Kombinierbarkeit von Managed und Unmanaged Code in C++/CLI stärker betont.

Große Teile der verwendeten Softwareprodukte waren nicht in der Lage, Echtzeitfähigkeit zu gewährleisten. So bietet Microsofts .NET-Plattform standardmäßig keine Echtzeitfähigkeit. Auch die eingesetzte Linux Distribution Ubuntu 10.04 verwendet keinen Echtzeitkernel. Deshalb mussten zwischen den Systemkomponenten sehr viele logische Verbindungen hergestellt werden, über die Signale verschickt wurden, die den Abschluss bestimmter Operationen und Berechnungen signalisieren.

Durch die strikte Trennung von Signalverarbeitung und Robotersteuerung können Erweiterungen bezüglich Bildverarbeitung oder Optimierung der Trajektorien vom Entwickler einfach an den entsprechenden Stellen eingepflegt werden, ohne dass dieser Modifikationen an anderen Systemkomponenten vornehmen muss. Dennoch bietet die lose Kopplung der Komponenten auch Nachteile. Trotz Bemühungen die Komponenten so zu koppeln, dass eine robuste Kommunikation zu Stande kommt, kann es in seltenen Fällen vorkommen, dass Transformationen falsch im Volume Builder Modul ankommen und so die Datensätze nicht korrekt berechnet werden können. Diese Fehler sind Resultat von Synchronisations- und Übertragungsproblemen, die schwerlich komplett ausgeschlossen werden können. Diese Fehler traten während des Einsatzes des Tomographen nur in den seltensten Fällen auf und waren jederzeit deutlich zu erkennen. Die fehlerhaften Datensätze enthielten keinerlei Nutzinformation bzw. hatten eine Ausdehnung von 0 auf der Z-Achse.

Außerdem müssen beim Start des Ultraschalltomographen die Routings zwischen den Systemkomponenten aufwändig vom Nutzer erstellt werden, damit die Anwendung vollständig einsatzbereit ist. Dieses Tribut an die lose Kopplung der Komponenten erhöht die Komplexität des Ultraschalltomographen hinsichtlich der Anwendung.

Matlab erwies sich zur Steuerung der Komponenten als hervorragend geeignet. Insbesondere die Berechnung der Transformationen zur Steuerung des Roboters und zum Versand an 3D Slicer kann mittels Matlab deutlich einfacher realisiert werden, als dies in Hochsprachen wie C++, oder Java der Fall ist. Matlab zeigt außerdem in seiner Kommandozeile jederzeit auf Wunsch die gerade ausgeführte Berechnung an, so dass die korrekte Funktion des Tomographen in Echtzeit nachvollzogen werden kann.

Obwohl die einzelnen Komponenten schlank gehalten wurden, entstand aufgrund der Verwendung verschiedenster Programmiersprachen und Hardwarekomponenten ein hochkomplexes Gesamtsystem. Gerade durch die lose Kopplung der Komponenten wurde ein großer Kommunikationsoverhead erzeugt, der bei Integration aller Anwendungsteile in ein einziges Gerät minimiert werden kann. Zur Entwicklung eines praxistauglichen Ultraschalltomographen ist das hier verwendete Konzept aufgrund der komplexen Bedienung ungeeignet. Es bietet Entwicklern die Möglichkeit, flexibel in einzelne Schritte des Prozesses einzugreifen und diese zu optimieren bzw. eigene Erweiterungen zu integrieren, ist im Vergleich zu einem System, das alle Komponenten des Ultraschalltomographen in ein einzelnes Soft-/Hardwareprodukt integriert, in der Anwendung weniger robust und vor allem deutlich langsamer. Gerade im klinischen Alltag ist große Geschwindigkeit zwecks Steigerung der Produktivität und der Bildqualität ein Vorteil. Das entwickelte System benötigt zur Aufnahme eines einzelnen Slices eine Dauer von circa 0.5-1.5 Sekunden. Zur Aufnahme von 200 Schichtaufnahmen wird also bereits eine Zeit von 100-300 Sekunden Dauer beansprucht. Es hat sich gezeigt, dass es sehr schwierig für Probanden ist, den Arm über einen längeren Zeitraum hinweg nicht zu bewegen. Die Vermutung liegt nahe, dass dieses Problem vor allem bei der Verwendung im Bereich des Thorax und Abdomens verstärkt zu Tage tritt, da in diesen Bereichen Bewegungen durch die Atmung des Probanden unvermeidbar sind.

Zur Entwicklung eines praxistauglichen Ultraschalltomographen wäre es also empfehlenswert, die Komponenten eng zu koppeln. Die Steuerung aller Modalitäten sowie die Generierung der Bilddatensätze sollten in einem einzelnen Softwareprodukt ablaufen. Idealerweise sollten echtzeitfähige Software- und Hardwareprodukte verwendet werden. 3D Slicer bietet sich auch als Entwicklungsbasis für ein System an, das alle Komponenten des Ultraschalltomographen in ein einzelnes Soft-/Hardwareprodukt integriert. Das DiPhAS API kann mittels C++/CLI direkt in ein Slicer Modul integriert werden. Der Aufwand hierzu ist allerdings ungleich größer als bei Verwendung von DiPhAS Networker über OpenIGTLink. Ebenso kann die Steuerung des Roboters in einem eigenen 3D Slicer Modul erfolgen. Leider existiert für 3D Slicer keinerlei medizinische Zulassung, so dass es notwendig wäre ein geeignetes Softwareprodukt nach den gültigen Gesetzen und Vorgaben der Medizin zu entwickeln, sollte es gewünscht sein den Ultraschalltomographen klinisch anzuwenden.

All diese Anforderungen wurden an das hier vorgestellte Gerät nicht gestellt. Es sollte eine flexible, leicht erweiterbare, robuste Entwicklungsumgebung für Methoden und Verfahren in der Ultraschalltomographie entwickelt werden, die in der Lage ist 3D Datensätze von Phantomen und Weichgewebe zu erstellen. Mit dem vorgestellten Ultraschalltomographen wurden diese Ziele erreicht. Damit bietet dieser die Möglichkeit, die eingesetzten Verfahren und Algorithmen weiterzuentwickeln und zu verbessern. Die Erkenntnisse aus dieser Arbeit können als Entwicklunsgbasis für ein praxistaugliches Gerät dienen.

Die gewonnen Ergebnisse zeigen eine Auflösung des Tomographen, die in allen Richtungen mindestens der lateralen Auflösung der Ultraschallsonde entspricht. Außerdem können wie bereits in Kapitel 6 erwähnt sehr geringe Winkelfehler erreicht werden. Die Bildpunkte innerhalb der generierten 3D-Datensätze können somit im Vergleich zu Verfahren, die mittels Speckle-Dekorrelation arbeiten, wesentlich genauer kartesischen Positionen im Körper zugeordnet werden. Mittels der vorhandenen Werkzeuge bestand keinerlei Möglichkeit, die Datensätze auf Submillimeter Ebene bezüglich ihrer absoluten Ortsauflösung zu validieren. Da allerdings die Korrektheit der Datensätze evaluiert werden konnte, muss davon ausgegangen werden, dass Fehler auf der Z-Achse der Datensätze lediglich durch Roboterungenauigkeit und durch die Stärke des Ultraschallstrahls verursacht werden. Die Präzision aktueller optischer Trackingsysteme von NDI liegt unter Optimalbedingungen im Bereich von 0.25 mm[48]. Die Wiederholgenauigkeit der verwendeten Roboter liegt im Bereich von 0.02-0.05 mm. Die Vermutung liegt deshalb nahe, dass mittels robotergestützter Ultraschalltomographie präzisere 3D-Datensätze erstellt werden können, als dies mittels optischer oder elektromagnetischer Trackingsysteme der Fall ist. Insbesondere bei Verwendung großer Eindringtiefen dürfte sich dies aufgrund geringer Winkelfehler bei der Ultraschalltomographie bemerkbar machen. Genauere Untersuchungen bezüglich der absoluten Genauigkeit könnten Gegenstand weiterer Arbeiten in Zusammenhang mit dem entwickelten Ultraschalltomographen bilden.

Bezüglich der Flexibilität der Handhabung ist der entwickelte Ultraschalltomograph Freihand-3D-Ultraschallverfahren unterlegen. Der Anwender kann die Sonde an interessanten Strukturen lediglich in kleinen Abständen verfahren und den Aufnahmewinkel ändern. Die Darstellung kann allerdings nicht mittels Auge-Hand-Koordination verbessert werden, wie das bei Freihand-Verfahren der Fall ist. Die gesammelten Datensätze zeigen, dass sich mittels der robotergestützten Ultraschalltomographie die Vorteile von Verfahren, die mit 2D-Arraysonden arbeiten und Verfahren die mittels externer Trackingsysteme arbeiten, vereinen lassen. Der Ultraschalltomograph weist einerseits den großen Arbeitsbereich auf, der bisher extern getrackten Verfahren vorbehalten war, und liefert andererseits Datensätze, in denen jedes Slice mit konstanter Orientierung der Sonde aufgezeichnet wurde, wie dies bei 2D-Arraysonden der Fall ist. Außerdem unterliegt robotergestützte Ultraschalltomographie anders als dies bei optisch getrackten System der Fall ist keinerlei Abhängigkeiten bezüglich Beleuchtung und Aufstellungsort.

Das System bietet vielfältige Möglichkeiten zur Entwicklung von Verfahren für die Ultraschalltomographie. Besonders bei der Nachverarbeitung der Datensätze besteht hierbei Potential. Interessante Ansätze können bezüglich der Verbesserung der Ortsauflösung des Systems verfolgt werden. Werden wie in Abbildung 54 gezeigt mehrere Datensätze verwendet, die

[48] http://www.ndigital.com/medical/polarisfamily-techspecs.php (letzter Zugriff: 01.12.2010)

gegeneinander registriert sind und dasselbe Volumen zeigen, besteht die Möglichkeit diese miteinander zu verrechnen. Bei der Verwendung vieler verschiedener Aufnahmewinkel, die möglicherweise sogar den kompletten Bereich von 0-360 Grad um das Volumen abdecken, können diese z.B. mittels linearer Interpolation zu einem einzigen Datensatz verrechnet werden. Denkbar wäre es dabei, die Datensätze so gegeneinander zu gewichten, dass die geringere laterale Auflösung der Sonde weniger ins Gewicht fällt und so eine allgemein deutlich höhere Ortsauflösung erreicht werden kann. Dies betrifft nicht nur den 3D-Datensatz als ganzes, sondern könnte auch die Ortsauflösung der 2D-Slices verbessern (Kombination mehrerer 2D Aufnahmen in derselben Ebene, aber mit unterschiedlichen Orientierungen der Sonde).

Des Weiteren bietet sich die Möglichkeit, die Sonde nicht ausschließlich auf ihrer Z-Achse zu verfahren, sondern mehrere Scans durchzuführen, bei der die Sonde auf ihrer X-Achse über das Volumen verfahren wird. Hierbei wäre es möglich nach jedem Scan auf der X-Achse die Sonde um den gewünschten Schichtabstand auf ihrer Z-Achse zu verschieben um ein 3D-Volumen zu erzeugen. Die entstandenen Datensätze könnten nach einer angepassten Nachverarbeitung in Richtung der X-Achse der Sonde (laterale Auflösung) eine verbesserte Ortsäuflösung zeigen.

Wird nicht ausschließlich der B-Mode des Ultraschallgerätes verwendet, sondern außerdem der A-Mode und werden Scans aus verschiedenen Orientierungen erstellt, besteht unter Umständen die Möglichkeit, aufgrund der geometrischen Verhältnisse die Schallgeschwindigkeit in allen Teilen des Volumens zu berechnen und damit geometrische Verzerrungen durch Geschwindigkeitsunterschiede auszugleichen.Bei Verwendung des Dopplermodus oder des M-Modes des Ultraschallgerätes bestände die Möglichkeit, den Blutfluss in Arterien oder die Bewegung der Herzklappen in 3D darzustellen.

Schließlich könnte dem Anwender auch noch die Verwendung des Roboterarmes als mechanisches Trackingsystem ermöglicht werden. So könnte zuerst ein Scan des Gewebes mittels robotergestützer Ultraschalltomographie durchgeführt werden und anschließend mittels Auge-Hand-Koordination die Sonde über besonderes interessante Bereiche im Gewebe bewegt werden. Die gewonnenen Datensätze können verrechnet und für die Darstellung präziserer Datensätze verwendet werden.

Alle gesammelten Datensätze sind außerdem ungefiltert und unsegmentiert. Es besteht zur Verbesserung der Bildqualität die Möglichkeit, das Speckle-Rauschen z.B. mittels Filtern auf Basis der Waveletanalyse zu entfernen. Außerdem existieren Verfahren zur Segmentierung von Knochenoberflächen in Ultraschallbildern, die ebenfalls auf die Datensätze angewendet werden können. Werden ein Verfahren zur Segmentierung von Knochen und das Verrechnen mehrerer Datensätze aus verschiedenen Orientierungen kombiniert, besteht unter Umständen die Möglichkeit, Knochenoberflächen im Körper des Probanden von allen Seiten in 3D darzustellen, wie das z.B. bei CT-Datensätzen der Fall ist . Mit konventionellen Verfahren kann die distal der Sonde liegende Kante eines Knochens wegen Schatten, die von der proximal der Sonde liegenden Kante verursacht werden, kaum bis überhaupt nicht dargestellt werden.

Insgesamt entstand ein System zur Entwicklung von Verfahren für die noch sehr junge Ultraschalltomographie, das sehr viel Potential für die Verbesserung der Bildqualität und der Handhabung von Ultraschallmodalitäten in der Medizin bietet. Es ist zu vermuten, dass die vorgestellten Konzepte und Algorithmen auch in anderen Einsatzbereichen wie z.B der Materialprüfung von Nutzen sein könnten.

8. Anhang

A: Quellcode des Algorithmus zum Entfernen der Bildränder (C#)

```
public void image_delegate(Bitmap bmp, int XDC_number, bool
represents_filtered_data)
{
      //if all OpenIGTLink Ports are connected
      if (imagePortConnected && firstPackageReceived && imageServerStarted)
      {
            //show Image on GUI
            this.imageDelegate(bmp);

            //START accessing bitmap data
            BitmapData bData = bmp.LockBits(new Rectangle(new Point(),
            bmp.Size),ImageLockMode.ReadOnly,PixelFormat.Format24bppRgb);
            int byteCount = bData.Stride * bmp.Height;
            byte[] bmpBytes = new byte[byteCount];
            //Copy all Bytes of Bitmap into a new array
            Marshal.Copy(bData.Scan0, bmpBytes, 0, byteCount);
            bmp.UnlockBits(bData);
            //STOP accessing bitmap data

            //Convert the bitmap to greyscale vector
            byte[] greyscale = new byte[bmp.Width * bmp.Height];
            greyscale = this.convertToGreyscale(bmpBytes, bmp.Width,
            bmp.Height);



            //If the Parameters of DiPhAS has recently changed
            if (this.parametersChanged)
            {
                  //receive 60 images in order to give DiPhAS enough time
                  to reconfigure
                  imagesAfterChange++;
                  if (this.imagesAfterChange == 60)
                  {
                        //set the First Black Pixel to the last pixel in
                        each line
                        int horizontalFirstBlack = bmp.Width - 1;
                        //walk through the first line of the image until
                        the center is reached (right to left)
                        for (int i = bmp.Width - 1; i >= bmp.Width / 2; i-)
                        {
                              //print the greyvalue on the console (debug
                              information)
                              System.Console.WriteLine("i " + i + "
                              greyscale " + greyscale[i + 10 * bmp.Width]);



                              //use the tenth line of image to check the
                              greyscale information. if value is 0 the
                              artifacts have not been discovered
                              if (greyscale[i + 10 * bmp.Width] == 0)
                              {
```

```
                                    //in this case decrease the index of the
                                    first pixel that is black
                                    horizontalFirstBlack = i;
                              }
                        }
                        //at this point the first pixel right of the center
                        of the image in the tenth line of the image is
                        discovered.
                        //the borders of the image and the black areas are
                        discovered
                        minimumX1 = 0;
                        maximumX1 = bmp.Width - 1 - horizontalFirstBlack;
                        minimumX2 = horizontalFirstBlack;
                        maximumX2 = bmp.Width - 1;
                        //show the borders on the Console
                        System.Console.WriteLine("minimumX1 " + minimumX1 +
                        " minimumX2 " + minimumX2 + " maximumX1 " +
                        maximumX1 + " maximumX2 " + maximumX2);
                        //reset the parameters for the next reconfiguration
                        of DiPhAS
                        parametersChanged = false;
                        imagesAfterChange = 0;
                  }

            }
            //now compute the image to send via OpenIGTLink
            //allocate an array that is big enough to hold the data of the
            image in each case
            byte[] finalImage= new byte[bmp.Height * bmp.Width];
            //if the borders have been calculated
            if (!this.parametersChanged)
            {
                  int counter = 0;

                  //for all the lines of the image
                  for (int v = 0; v < bmp.Height; v++)
                  {
                        //for all pixels of a line in the area of the
                        Ultrasound Image (without black areas)
                        for (int u = maximumX1 + 1; u < minimumX2; u++)
                        {
                              //insert the current pixel into the array that
                              holds the information to send via openIGTLink
                              finalImage[counter] = greyscale[u + v *
                              bmp.Width];
                              //increase the counter for the current
                              position in the array
                              counter++;
                        }
                  }
            }




            //Send the image via the link
            //if DiPhAS is configured and borders have bee claculated
            if (!this.parametersChanged)
            {
                  this.letztesBitmap = greyscale;
```

```
                    //if an image has been requested by Slicer
                    if (sendImage)
                    {
                                //Output the Width of the Ultrasound
                                information on the Console. (debug
                                information)
                                System.Console.WriteLine("XDim " + (minimumX2
                                - maximumX1 - 1));
                                //set the dimensions for the IMAGE message
                                imageServer.setDimensions((minimumX2 -
                                maximumX1 - 1), imgY, 1);
                                //set the spacings to the current image
                                spacings calculated by DiPhAS
                                imageServer.setSpacing(this.depth * 1000.0f /
                                (float)imgX, this.depth * 1000.0f /
                                (float)imgY, 1.0f);
                                //Display the spacings on the Console (debug
                                information)
                                System.Console.WriteLine(this.depth * 1000.0f
                                / (float)imgX);
                                imageServer.setScalarTypeToUINT8();
                                imageServer.setDeviceName("DiPhAS");
                                //set a dummy matrix
                                imageServer.setOrientationMatrix(0.0f, 0.0f, -
                                1.0f, 0.0f, -1.0f, 0.0f, 0.0f, 0.0f, 0.0f, -
                                1.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 1.0f);
                                //allocate the array in the IMAGE message
                                imageServer.allocateScalars();
                                imageServer.setSubVolume((minimumX2 -
                                maximumX1 - 1), imgY, 1, 0, 0, 0);
                                //copy the image information into the IMAGE
                                message array
                                imageServer.loadData(finalImage);
                                //pack the message
                                imageServer.pack();
                                //send the message via OpenIGTLink
                                int success = imageServer.send();
                                //if the image could not be successfully send
                                reconnect to the client
                                if (success != 1)
                                {
                                      System.Console.WriteLine("Reconnect");
                                      reconnectImageServer();

                                }
                                else
                                {
                                      System.Console.WriteLine("imageSend");
                                }
                          //image has been send. wait for next request
                          sendImage = false;
                    }
              }


      }
}
/**
 * converts an RGB vector with given size into a greyscale vector
 * @param data the greyscale vector. size is 3*x*y
 * @param x width of the image the vectors contains to
 * @param y height of the image the vectors contains to
```

```
 * @return the greyscale vector with lengt x*y
 * */
public byte[] convertToGreyscale(byte[] data,int x,int y)
{
	//allocate the array with greyscale information
	byte[] greyscale = new byte[x * y];
	int byteCounter = 0;
	//for each pixel remove 2 of three channels
	for (int i = 0; i < x * y; i++)
	{
		//use the current channel of the pixel
		greyscale[i] = data[byteCounter];
		//increase the counter for the position in the array by three
		to go to the first channel of the next pixel
		byteCounter += 3;
	}
	return greyscale;
}
```

B: Quellcode des Algorithmus zum Entfernen der Ankopplungsartefakte (C++)

```
void vtkVolumeBuilderModuleLogic::cropBorder() {
	vtkMRMLScalarVolumeNode * currentVolumeNode
	//load the first image of the image series. gets the image out of the
	id held by the list that contains all scan information
	currentVolumeNode = vtkMRMLScalarVolumeNode::SafeDownCast(
			this->MRMLScene->GetNodeByID(
				this->VolumeBuilderModuleNode->getImage(0)));
	int size[3] = { 0, 0, 0 };
	//get the dimensions of the images in the series ( no image in the
	list may be bigger than the first one)
	currentVolumeNode->GetImageData()->GetDimensions(size);
	int dims[3];
	//dynamically allocate space for the image data
	void *valsVoid = malloc(size[0] * size[1] * +size[2] * sizeof(char));
	char * vals = (char*) valsVoid;
	//set each value in the freshly allocated space to value 0 (greyscale
	black)
	for (int j = 0; j < size[0] * size[1] * size[2]; j++) {
		vals[j] = 0;
	}
	int currentDepth;
	char *currentPtr;
	//walk through the list that contains the scan informations
	for (int i = 0; i < this->VolumeBuilderModuleNode->
	getSizeOfNodeList(); i++) {
		//get the next imgae of the series
		currentVolumeNode = vtkMRMLScalarVolumeNode::SafeDownCast(
				this->MRMLScene->GetNodeByID(
				this->VolumeBuilderModuleNode->getImage(i)));
		//get the current entrydepth for the current image

		currentDepth
				= vtkMRMLDiphasParameterModuleNode::SafeDownCast(
					this->MRMLScene->GetNodeByID(this->
		VolumeBuilderModuleNode->getParameter(i)))->GetEntryDepth();
		//get the dimensions of the current image
		currentVolumeNode->GetImageData()->GetDimensions(dims);
		//calculate the mm/pixel in order to get the number of lines
		that have to be colored (black)
```

```
            double mmPerPixel = (double) currentDepth / (double) dims[1];
            //calculate the numbers of lines to set to black
            double pixDouble = border / mmPerPixel;
            //round the value
            int pix = std::ceil(pixDouble);
            //retrieve the pointer taht points on image data
            currentPtr = (char*) currentVolumeNode->GetImageData()->
            GetScalarPointer();
            //copy black values at the correct position in the array that
            contains only ultrasound information. Afterwards the artifacts
            at the top of the images are colored black
            memcpy(currentPtr, vals, pix * dims[1] * sizeof(char));
      }
      //free the dynamically alocated space
      free(valsVoid);
}
```

C: Quellcode des Algorithmus zum Einpassen der Bilder in Volumendatensätze

```
void vtkVolumeBuilderModuleLogic::adjustTrace() {
      void* blackVoid;
      char* black;

      //Maximalwerte fuer Y
      double yMin = 0;
      double yMax = 0;
      //Referenzwert fuer Ebene
      double xRef = 0;
      double translate[3] = { 0, 0, 0 };
      vtkMatrix4x4 * imageTransform;
      vtkMatrix4x4 * referenceTransform = vtkMatrix4x4::New();
      bool alreadyUsed = false;
      bool alreadyUsedInside = false;
      int i = 0;
      int size[3] = { 600, 600, 1 };
      double translateForVolume[3] = { 0, 0, 0 };
      std::vector<char *> returnWorkingSet;

      //the entrydepth doesn't change during a scan. so it can be retrieved
      at the beginning of this algorithm
      vtkMRMLDiphasParameterModuleNode * parameters =
                  vtkMRMLDiphasParameterModuleNode::SafeDownCast(
                  this->MRMLScene->GetNodeByID(this->
                  VolumeBuilderModuleNode->getParameter(0)));
      int entrydepth = parameters->GetEntryDepth();
      //output the entrypdepth on the Console (debug information)
      std::cerr << "Eindringtiefe" << entrydepth << std::endl;
      //same with spacings
      vtkMRMLScalarVolumeNode * image =
      vtkMRMLScalarVolumeNode::SafeDownCast(
                  this->MRMLScene->GetNodeByID(
                              this->VolumeBuilderModuleNode->getImage(0)));
      image->GetImageData()->GetDimensions(size);
      double spacings[3];
      image->GetSpacing(spacings);
      //create a vector that holds all the allready used nodes
      std::vector<char *> alreadySearchedNodes;
      //the current working set that holds all the id of the images of a
      series
      std::vector<char *> workingSet;
```

```
//walk through the transformationlist to identify all the datasets of
one series
std::cerr << "Durchlaufe Liste" << std::endl;
while (i < this->VolumeBuilderModuleNode->getSizeOfNodeList()) {
      //start a new series and a ne workingSet
      std::cerr << "neues Volumen " << std::endl;
      workingSet.clear();
      returnWorkingSet.clear();
      //retrieve the first transform of the series as reference for
      all following images
      char
                  * transform1ID =
                              vtkMRMLScalarVolumeNode::SafeDownCast(
                                          this->MRMLScene->
                                          GetNodeByID(this->
                                          VolumeBuilderModuleNode->
                                          getImage(i)))->
                                          GetTransformNodeID();

      //check if the transformationNode has allready be used
      for (int h = 0; h < alreadySearchedNodes.size(); h++) {
            if (transform1ID == alreadySearchedNodes.at(h)) {
                  alreadyUsed = true;
            }
      }
      std::cerr << "Ueberpruefe 1 Knoten auf Benutzung" << std::endl;
      //if the transformation has not been used use it as reference
      for the new series
      if (!alreadyUsed) {
            std::cerr << "Knoten bisher unbenutzt" << std::endl;
            alreadySearchedNodes.push_back(transform1ID);
            //add the first image node to the workingSet
            workingSet.push_back(this->VolumeBuilderModuleNode->
            getImage(i));
            //setting up the reference transform
            imageTransform
                        = vtkMRMLLinearTransformNode::SafeDownCast(
                                    this->MRMLScene->
                                    GetNodeByID(transform1ID))->
                                    GetMatrixTransformToParent();
            for (int u = 0; u < 4; u++) {
                  for (int v = 0; v < 4; v++) {
                        referenceTransform->SetElement(u, v,
                                    imageTransform->GetElement(u, v));
                  }
            }
            translateForVolume[0] = imageTransform->GetElement(0, 3);
            translateForVolume[2] = imageTransform->GetElement(2, 3);
            //display the inital transform for the series
            std::cerr << "Initiale Endtranslationen" <<
            translateForVolume[0]
                        << " " << translateForVolume[1] << " "
                        << translateForVolume[2] << std::endl;
            //calculate the constant value on x axis of the probe
            //the coordinates in list are coordiantes of the robot
            base, so the base needs to be changed by using -
            transpose®*t
            this->computeMinusRTransMalT(imageTransform, translate);
            //set the reference
            xRef = translate[0];

            //initialize borders
```

```
yMin = translate[1];
yMax = translate[1];

std::cerr << "Translation in Bildkoordinaten" <<
translate[0]<< " " << translate[1] << " " << translate[2]
<< std::endl;
std::cerr << "Initialwerte fuer Grenzen" << std::endl;
//walk through the whole list to calculate the borders
std::cerr << "Bestimme Grenzwerte" << std::endl;
for (int k = i + 1; k
            < this->VolumeBuilderModuleNode->
            getSizeOfNodeList(); k++) {
      std::cerr << "Schrittweise Knoten ueberpruefen" <<
      std::endl;
      char  * transformToCheckID =
            vtkMRMLScalarVolumeNode::SafeDownCast(
            this->MRMLScene->GetNodeByID(

      this->VolumeBuilderModuleNode->getImage(k)))->
      GetTransformNodeID();
      //check if the node has been used before
      std::cerr << "Ueberpruefe Knoten auf Benutzung
      Inside"<< std::endl;
      for (int h = 0; h < alreadySearchedNodes.size();
      h++) {
            if (transformToCheckID ==
            alreadySearchedNodes.at(h)) {
                  alreadyUsedInside = true;
            }
      }
      // if the node has not been used
      if (!alreadyUsedInside) {
            std::cerr << "Knoten Inside Unbenutzt" <<
            std::endl;
            imageTransform =
            vtkMRMLLinearTransformNode::SafeDownCast(this-
            >MRMLScene->GetNodeByID(
            transformToCheckID))->
            GetMatrixTransformToParent();
            //calculate the x Value in Probe coordinates
            in order to compare to the reference
            this->computeMinusRTransMalT(imageTransform,
            translate);
            //add a small tolerance for numeric and robot
            errors
            double tolerance = 1;
            std::cerr << (xRef - tolerance) << " " <<
            translate[0]
                        << " " << (xRef + tolerance) <<
            std::endl;
            //check if images are of the same series
            if (translate[0] >= (xRef - tolerance) &&
            translate[0]<= (xRef + tolerance)
                        && referenceTransform->
                        GetElement(0, 0)
                              == imageTransform->
                        GetElement(0, 0)
                        && referenceTransform->
                        GetElement(0, 1)
                              == imageTransform->
                        GetElement(0, 1)
```

```
            && referenceTransform->
            GetElement(0, 2)
                        == imageTransform->
            GetElement(0, 2)
            && referenceTransform->
            GetElement(1, 0)
                        == imageTransform->
            GetElement(1, 0)
            && referenceTransform->
            GetElement(1, 1)
                        == imageTransform->
            GetElement(1, 1)
            && referenceTransform->
            GetElement(1, 2)
                        == imageTransform->
            GetElement(1, 2)
            && referenceTransform->
            GetElement(2, 0)
                        == imageTransform->
            GetElement(2, 0)
            && referenceTransform->
            GetElement(2, 1)
                        == imageTransform->
            GetElement(2, 1)
            && referenceTransform->
            GetElement(2, 2)
                        == imageTransform->
            GetElement(2, 2)) {
      //the node will be used, so add to
      allready used nodes

      alreadySearchedNodes.
      push_back(transformToCheckID);
      //add node to working set
      workingSet.push_back(this->
      VolumeBuilderModuleNode->getImage(k));
      //calculate the new borders on y axis of
      the probe for the volume
      if (translate[1] < yMin) {
            yMin = translate[1];
            std::cerr << "neue obere Grenze"
            << std::endl;
      } else if (translate[1] > yMax) {
            yMax = translate[1];
            //if there is new upper border the
            transform for the whole volume has
            to be refreshed
            translateForVolume[0] =
            imageTransform->GetElement(
                        0, 3);
            translateForVolume[2] =
            imageTransform->GetElement(
                        2, 3);
            std::cerr << "Neue
            Endtranslationen"
            << translateForVolume[0] << " "
            << translateForVolume[1] << " "
            << translateForVolume[2] <<
            std::endl;
            std::cerr << "neue untere Grenze"
            << std::endl;
      }
```

```
                }
                std::cerr << "Neue Grenzen: " << yMin << " "
                << yMax
                << std::endl;
        }
        alreadyUsedInside = false;
}
//now the borders are know. display all the informations
on Console (debug information)
std::cerr << "finale Grenzen " << yMin << " " << yMax <<
std::endl;
std::cerr << "spacings" << spacings[0]<< " " <<
spacings[1]<<" " << spacings[2]<< std::endl;
std::cerr << "entrydepth" << entrydepth << std::endl;
//calculate the size of the Volume on Y axis of the probe
int ySizeNew = ((yMax - yMin) / spacings[1]) + 600;
//dynamically allocate space for the new image
information
blackVoid = malloc(size[0] * ySizeNew * size[2] *
sizeof(char));
black = (char*) blackVoid;
//fill up the new array with greyscale value black (0)
for (int v = 0; v < size[0] * ySizeNew * size[2]; v++) {
        black[v] = 0;
}
std::cerr << "Distanz zwischen Grenzen in Pixeln" <<
ySizeNew
<< std::endl;
int newDimensions[3] = { size[0], ySizeNew, 1 };
//walk through the workingSet in order to add black
borders at the beginning and at the ending of the image
array for the reason of using the same y and x of
transform for each image in the series
if (workingSet.size() > 0) {
        //get the transform of the first image in
        workingSet
        vtkMRMLScalarVolumeNode
                        * volumeRef =
                        vtkMRMLScalarVolumeNode::SafeDownCast(
                                        this->MRMLScene->
                                        GetNodeByID(
                                        workingSet.front()));
        vtkMRMLLinearTransformNode * transformRef =
                vtkMRMLLinearTransformNode::SafeDownCast(
                this->MRMLScene->GetNodeByID(
                                volumeRef->GetTransformNodeID()));

        //fill up the array of each image with the black
        borders
        while (workingSet.size() > 0) {
                //retrieve the image


                vtkMRMLScalarVolumeNode * volumeToModify =
                        vtkMRMLScalarVolumeNode::SafeDownCast(
                                        this->MRMLScene->
                                        GetNodeByID(
                                        workingSet.back()));
                //remove the image from working
                workingSet.pop_back();
```

```
vtkMRMLLinearTransformNode    * transform =
vtkMRMLLinearTransformNode::SafeDownCast(
            this->MRMLScene->GetNodeByID(
            volumeToModify->
            GetTransformNodeID()));
imageTransform = transform->
GetMatrixTransformToParent();
//calculate transform in probe coordiantes
this->computeMinusRTransMalT(imageTransform,
translate);
//calculates how many lines of black has to be
inserted on top of information
double distanceTop=yMax-translate[1];
int numberOfPixelsImageYTop = distanceTop /
spacings[1];

//create a new image data for the node
std::cerr << "working 1" << std::endl;
vtkImageData * oldImageData =
            volumeToModify->GetImageData();
vtkImageData * newImageData =
vtkImageData::New();
newImageData->SetDimensions(newDimensions[0],
newDimensions[1], newDimensions[2]);
newImageData->SetExtent(0, newDimensions[0] -
1, 0, newDimensions[1] - 1, 0,newDimensions[2]
- 1);
newImageData->SetOrigin(0.0, 0.0, 0.0);
newImageData->SetSpacing(1.0, 1.0, 1.0);
std::cerr << "working 2" << std::endl;
volumeToModify->SetSpacing(spacings[0],
spacings[1], 1.0);
newImageData->SetNumberOfScalarComponents(1);
newImageData->SetScalarTypeToUnsignedChar();
newImageData->AllocateScalars();
char * oldD = (char*) oldImageData->
      GetScalarPointer();
char * newD = (char*) newImageData->
GetScalarPointer();
std::cerr << "working 3" << std::endl;
//copy black into the array of the new
ImageData
memcpy(newD, blackVoid, size[0] * ySizeNew *
size[2]* sizeof(char));
//copy the ultrasound image to the correct
position in the array
memcpy(newD + size[0] *
numberOfPixelsImageYTop, oldD,
size[0] * size[1] * size[2] * sizeof(char));
//set the new ImageData to the old node
volumeToModify->
SetAndObserveImageData(newImageData);


//set the transform in a way that all
transforms of all the images only vary on the
y-Axis of the probe
imageTransform->SetElement(0, 3,
translateForVolume[0]);
imageTransform->SetElement(2, 3,
translateForVolume[2]);
std::cerr << "gesetzte Transformation"
```

```
                                    << translateForVolume[0] << " "
                                    << imageTransform->GetElement(1, 3) << "
                                    "<< translateForVolume[2] << std::endl;
                                std::cerr << "finish" << std::endl;

                        }
                    }
                }
                alreadyUsed = false;
                i++;

        }
        free(blackVoid);

}
```

D: Quellcode des Algorithmus zur Generierung der Volumendatensätze

```
void vtkVolumeBuilderModuleLogic::buildVolume() {
        std::cerr << "Split Volumes" << std::endl;
        vtkMRMLScalarVolumeNode * outNode;
        vtkMRMLScalarVolumeNode * currentVolumeNode;
        vtkMRMLLinearTransformNode * currentTransform;
        vtkImageData * imageData;
        outNode = vtkMRMLScalarVolumeNode::New();
        imageData = vtkImageData::New();
        int size[3] = { 600, 600, 1 };

        int sizeOfCurrentVolume = 0;
        int i = 0;
        int begin = 0;
        int end = 0;
        bool lookAtNext = true;
        double spacings[3];
        //walk through the list that holds the scan information
        while (i < this->VolumeBuilderModuleNode->getSizeOfNodeList()) {
                begin = i;
                end = i;
                //calculate the borders on probe z axis for the volume
                std::cerr << "Grenzen bestimmen" << std::endl;
                while (lookAtNext && i + 1
                        < this->VolumeBuilderModuleNode->
                        getSizeOfNodeList()) {
                        //retrieve the current and the following image in the
                        list
                        char* transform1ID =
                        vtkMRMLScalarVolumeNode::SafeDownCast(
                        this->MRMLScene->GetNodeByID(
                        this->VolumeBuilderModuleNode->getImage(i)))->
                        GetTransformNodeID();
                        char * transform2ID =
                        vtkMRMLScalarVolumeNode::SafeDownCast(
                        this->MRMLScene->GetNodeByID(
                        this->VolumeBuilderModuleNode->getImage(i+1)))->
                        GetTransformNodeID();
                        vtkMatrix4x4 * mat1 =
                        vtkMRMLLinearTransformNode::SafeDownCast(
                        this->MRMLScene->GetNodeByID(transform1ID))->
                        GetMatrixTransformToParent();
                        vtkMatrix4x4 * mat2 =
```

```
            vtkMRMLLinearTransformNode::SafeDownCast(
            this->MRMLScene->GetNodeByID(transform2ID))->
            GetMatrixTransformToParent();
            //check if the image are in the same series
            //if they are in the same series continue the check with
            each following imae
            if (mat1->GetElement(0, 0) == mat2->GetElement(0, 0)
                        && mat1->GetElement(0, 1) == mat2->
                        GetElement(0, 1)
                        && mat1->GetElement(0, 2) == mat2->
                        GetElement(0, 2)
                        && mat1->GetElement(1, 0) == mat2->
                        GetElement(1, 0)
                        && mat1->GetElement(1, 1) == mat2->
                        GetElement(1, 1)
                        && mat1->GetElement(1, 2) == mat2->
                        GetElement(1, 2)
                        && mat1->GetElement(2, 0) == mat2->
                        GetElement(2, 0)
                        && mat1->GetElement(2, 1) == mat2->
                        GetElement(2, 1)
                        && mat1->GetElement(2, 2) == mat2->
                        GetElement(2, 2)
                        && mat1->GetElement(0, 3) == mat2->
                        GetElement(0, 3)
                        && mat1->GetElement(2, 3) == mat2->
                        GetElement(2, 3)) {
                  i++;
                  end = i;
            } else {
                  lookAtNext = false;
            }
      }
      //build up the volume
      outNode = vtkMRMLScalarVolumeNode::New();
      imageData = vtkImageData::New();

      //get the last image of the series
      currentVolumeNode = vtkMRMLScalarVolumeNode::SafeDownCast(
      this->MRMLScene->GetNodeByID(
      this->VolumeBuilderModuleNode->getImage(end)));
      currentVolumeNode->GetImageData()->GetDimensions(size);
      //setup the image data of a new volume
      imageData->SetDimensions(size[0], size[1], end - begin + 1);
      imageData->SetOrigin(0.0, 0.0, 0.0);
      currentVolumeNode->GetSpacing(spacings);

      imageData->SetSpacing(1.0, 1.0, 1.0);
      imageData->SetExtent(0, size[0] - 1, 0, size[1] - 1, 0, end -
      begin);
      imageData->SetNumberOfScalarComponents(1);
      imageData->SetScalarTypeToUnsignedChar();
      //allocate space for the new volume
      imageData->AllocateScalars();

      //retrieve the pointer onto the new imageData
      char * imgPtr = (char*) imageData->GetScalarPointer();
      char * currentPtr;

      //create a new transform node
      vtkMRMLLinearTransformNode* newTransform =
                  vtkMRMLLinearTransformNode::New();
```

```
vtkMatrix4x4 * matrixIn = vtkMatrix4x4::New();
vtkMatrix4x4 * matrixOut = vtkMatrix4x4::New();

//retrieve the transformnode for the first image in series
matrixIn    = vtkMRMLLinearTransformNode::SafeDownCast(
this->MRMLScene->GetNodeByID(
vtkMRMLScalarVolumeNode::SafeDownCast(
this->MRMLScene->GetNodeByID(
this->VolumeBuilderModuleNode->getImage(begin)))->
GetTransformNodeID()))->GetMatrixTransformToParent();

matrixOut = newTransform->GetMatrixTransformToParent();
//copy the transform of the first image into the transformnode
of the new volume
memcpy(matrixOut, matrixIn, sizeof(vtkMatrix4x4));
//for each picture in the list: integrate the image into the
new volume
for (int j = 0; j <= end - begin; j++) {
      //if series contains more than one image
      if (j == 0 && j + 1 <= end - begin) {
            //calculate the spacing on z-axis by using the
            transforms of the first and the second imae
            char * transform1ID =
            vtkMRMLScalarVolumeNode::SafeDownCast(
            this->MRMLScene->GetNodeByID(
            this->VolumeBuilderModuleNode->getImage(j
            + begin)))->GetTransformNodeID();
            char * transform2ID =
            vtkMRMLScalarVolumeNode::SafeDownCast(
            this->MRMLScene->GetNodeByID(
            this->VolumeBuilderModuleNode->getImage(j
            + begin + 1)))->GetTransformNodeID();
            vtkMatrix4x4 * mat1 =
            vtkMRMLLinearTransformNode::SafeDownCast(
            this->MRMLScene->GetNodeByID(
            transform1ID))->GetMatrixTransformToParent();
            vtkMatrix4x4 * mat2 =
            vtkMRMLLinearTransformNode::SafeDownCast(
            this->MRMLScene->GetNodeByID(
            transform2ID))->GetMatrixTransformToParent();
            double spacingZ = mat2->GetElement(1, 3) - mat1->
            GetElement(1,3);
            //setup the spacings of the node
            outNode->SetSpacing(spacings[0], spacings[1],
            spacingZ);


      } else if (j == 0) {
            //if only one image is in series set the spacing on
            z axis to 1.0
            outNode->SetSpacing(spacings[0], spacings[1], 1.0);

      }
      //get the next image out of the list
      currentVolumeNode
      = vtkMRMLScalarVolumeNode::SafeDownCast(
      this->MRMLScene->GetNodeByID(
      this->VolumeBuilderModuleNode->getImage(j + begin)));
      currentPtr  = (char*) currentVolumeNode->GetImageData()->
      GetScalarPointer();
```

```
                //copy the ImageDate of a 2d image into the imageDate of
                the volume. Offset is used to insert the image at the
                correct position
                memcpy(imgPtr + size[0] * size[1] * j, currentPtr,
                currentVolumeNode->GetImageData()-> GetScalarSize()
                * size[0] * size[1]);
            }
            //Add the Nodes to the scene
            this->MRMLScene->AddNode(newTransform);
            outNode->SetAndObserveTransformNodeID(newTransform->GetID());
            outNode->SetAndObserveImageData(imageData);
            outNode->SetName("OutVolume");
            this->MRMLScene->AddNode(outNode);
            //create a display Node in order to show the image on the view
            vtkMRMLScalarVolumeDisplayNode *displayNode = NULL;
            displayNode = vtkMRMLScalarVolumeDisplayNode::New();
            displayNode->SetScene(this->MRMLScene);

            displayNode->SetLowerThreshold(0);
            displayNode->SetUpperThreshold(600);
            displayNode->SetWindow(600);
            displayNode->SetLevel(300);
            this->MRMLScene->AddNode(displayNode);
            //setup the algorithm to calculate the next series of the list
            i = end + 1;
            lookAtNext = true;

        }

}
```

E: Verwendete Software- und Hardwareprodukte

- **3D Slicer**
  Modulares, plattformunabhängiges OpenSource Softwarepaket für Visualisierung, Registrierung und Segmentierung medizinischer Datensätze. Entwickelt und gepflegt von der BWH Harvard und der Slicer Community: http://slicer.org

- **OpenIGTLink**
  Schlankes, leicht erweiterbares Netzwerkprotokoll für Anwendungen in der „Image guided therapie“. Es existieren Implementierungen für C, C++ und Matlab. Über OpenIGTLinkIF in 3D Slicer integriert. Entwickelt und gepflegt von der National Alliance for Medical Image Computing: http://www.na-mic.org/Wiki/index.php/OpenIGTLink

- **Eclipse**
  Universal einsetzbare plattformunabhängige OpenSource Entwicklungsumgebung für viele Programmiersprachen. Über Module erweiterbar. Entwickelt und gepflegt von der Eclipse Foundation: http://www.eclipse.org/

- **Visual Studio**
  Eine integrierte Entwicklungsumgebung der Firma Microsoft. Unterstützt werden vor allem die Sprachen der firmeneigenen .NET Familie: http://www.microsoft.com/visualstudio

- **MATLAB**
  Eine von Mathworks entwickelte Hochsprache und Entwicklungsumgebung, die vor allem für rechenintensive Aufgaben ausgelegt ist: http://www.mathworks.de/products/matlab/

- **VTK (Visualization Toolkit)**
  OpenSource Bibliothek für 3D Computergrafik, Visualisierung und Bildverarbeitung. Basiert auf OpenGL und enthält lange Zeit patentierte Algorithmen wie Marching Cube. Entwickelt und gepflegt von Kitware:
  http://www.vtk.org/

- **ITK (Insight Segmentation and Registration Toolkit)**
  OpenSource Bibliothek für Segmentierungs- und Registrierungsaufgaben und Aufgaben der medizinischen Bildverarbeitung. Entwickelt und gepflegt von Kitware:
  http://www.itk.org/

- **QT**
  Eine plattformunabhängige C++ Bibliothek für Oberflächen- und Applikationsentwicklung. Entwickelt und gepflegt von Nokia: http://qt.nokia.com/products/

- **KWWidgets**
  Eine freie plattformunabhängige C++ Bibliothek zur Entwicklung von Oberflächen. Entwickelt und gepflegt von Kitware: http://www.kwwidgets.org/Wiki/KWWidgets

- **MRML (Multimedia Retrieval Markup Language)**
  XML basiertes Datenformat zur einheitlichen Repräsentation von Multimediadaten: http://www.mrml.net/

- **AdeptV+**
  Programmiersprache und Betriebssystem für Anwendungen in der Robotik. Entwickelt und gefplegt von Adept: http://www.adept.de/produkte/software/embedded/v-plus/allgemeines

- **DiPhAS**
  Ein vom Fraunhofer Institut für biomedizinische Forschung entwickeltes Ultraschallsystem für Forschungszwecke. Das API bietet Zugang zu den meisten Signal- und Bildverarbeitungsschritten. DiPhAS hat keine medizinische Zulassung: http://www.ibmt.fraunhofer.de/fhg/ibmt/projekte/digitales_phased_array_system.jsp

- **LWR IV**
  Ein in Zusammenarbeit vom DLR und von KUKA entwickelter 7-Achs Knickarmroboter, der sich durch spezielle Kraftmomentensensorik, Gewicht und Anzahl der Freiheitsgrade von anderen Knickarmrobotern unterscheidet: http://www.kuka-robotics.com/germany/en/products/addons/lwr/start.htm

- **RX90**
  Ein von der Firma Stäubli entwickelter 6-Achs Knickarmroboter mit hoher Präzission. Ein Nachfolger wird unter dem selben Namen vertrieben: http://www.staeubli.de/

F: Inhaltsverzeichnis der beigelegten CD

Die beigelegte CD enthält den kompletten Quellcode, der während dieser Diplomarbeit entstanden ist. Alle verwendeten Online-Quellen wurden in PDF-Dateien konvertiert und sind auf der CD enthalten (Stand November 2010). Desweiteren sind die Ausschreibung zu dieser Diplomarbeit und die Arbeit selbst enthalten. Schließlich sind alle generierten Transformationen und Volumendatensätze, die auch im Text beschrieben sind auf der CD abgelegt, so dass diese jederzeit mit 3D Slicer begutachtet und bearbeitet werden können.

Die Verzeichnisstruktur auf der CD folgt hierbei folgendem Schema:

- Ausschreibung für diese Diplomarbeit (Diplomarbeit_Ausschreibung.pdf)
- Diplomarbeit (Diplomarbeit.pdf)
- Online-Quellen (Enthält alle Online-Quellen als PDF, oder PPT Dateien)
- Quellcode (enthält den Quellcode zu allen Systemkomponenten)
    - DiphasNetworker (enthält das Projekt DiphasNetworker zur Ausführung auf dem DiPhAS Ultraschallgerät)
    - DiphasOpenIGTLinkWrapper (enthält den Wrapper für DiphasNetworker, der benötigt wird um OpenIGTLink zu verwenden)
    - DiphasParameterModule (enthält das 3D Slicer Modul zur Konfiguration von DiPhAS)
    - MatlabIGTL_Linux (enthält die Matlab-OpenIGTLink Anbindung für Linux)
    - MatlabIGTL_Windows (enthält die Matlab-OpenIGTLink Anbindung für Windows)
    - OpenIGTLink_Linux (enthält die modifizierte OpenIGTLink Bibliothek für Linux)
    - OpenIGTLink_Windows (enthält die modifizierte OpenIGTLink Bibliothek für Windows)
    - robotControl (enthält den Matlab Skript und die GUI für robotControl)
    - RX90Control (enthält die um OpenIGTLink erweitere Steuerungssoftware für den RX90)
    - VolumeBuilderModule (enthält das 3D Slicer Modul, das die 2D-Ultraschallbilder sammelt und die 3D-Datensätze berechnet)
- Scans (Enthält die einzelnen während dieser Arbeit entstandenen Scans. Für jeden Scan sind Transformation, Volumendatensatz und eine TXT-Datei enthalten, in der die DiPhAS Konfiguration für diesen Scan hinterlegt ist)
    - Armscan_Oberseite (Scan der Oberseite des Armes)
    - Armscan_Unterseite (Scan der Unterseite des Armes)
    - Leberphantom (Scan des Leberphantoms)
    - Multimodalphantom (Scan einer Rippe im Multimodalphantom)
    - Rohr_liegend (Scan eines im Wasserbad liegenden Kunststoffrohres)
    - Rohr_stehend (Scan eines im Wasserbad stehenden Kunststoffrohres)

- Schaedel_Schneidezaehne-OsOccipitale (Scan des Schaedels von den Schneidezähnen bis zum Os Occipitale)
- Schlagstab (Scan des Schlagstabes eines Schlagzeugs)
- Jochbein_Schaedel (verschiedene Scans des Jochbeins eines Kunststoffschaedels)
    - 150 Grad (Scan aus Sonden-Winkel 150 Grad)
    - 180 Grad (Scan aus Sonden-Winkel 180 Grad)
    - 210 Grad (Scan aus Sonden-Winkel 210 Grad)

9. Literaturverzeichnis

[A. Gonzales et al.] A. Gonzales et al.: TER: a system for robotic tele-echography. Lecture Notes in Computer Science

[A. Lasso et al., 2010] Robot-assisted MRI-guided prostate biopsy using 3D Slicer, NA-MIC Tutorial Contest: Summer 2010

[B. Chiu et al., 2010] B. Chiu et al.: Three-Dimensional Carotid Ultrasound Segmentation Variability Dependence On Signal Difference And Boundary Orientation. Ultrasound in Med. & Biol., Vol. 36, No. 1, pp. 95–110, 2010

[BWH, 2010] Slicer 3 Extensions http://www.slicer.org/slicerWiki/index.php/Slicer3:Extensions (letzer Zugriff: 01.12.2010)

[BWH, 2010] Developing with Slicer3 http://www.slicer.org/slicerWiki/index.php/Slicer3:Developers (letzer Zugriff: 01.12.2010)

[BWH, 2010] Integrating with Slicer 3 http://www.slicer.org/slicerWiki/images/7/75/Integrating_with_Slicer3.ppt (letzer Zugriff: 01.12.2010)

[D. Napolitano, 2006] D. Napolitano.: Sound speed correction in ultrasound imaging. Ultrasonics 44 (2006) e43–e46

[D.Sutter, 2007] Medizinphysik Skript – Ultraschall, Dieter Suter 2007, Universität Dortmund

[DFG, 2010] MiroSurge http://www.dfg.de/download/pdf/dfg_magazin/wissenschaft_oeffentlichkeit/forschung_magazin/german_research_1_10_en.pdf (letzer Zugriff: 01.12.2010)

[DLR, 2010] Rollin‘ Justin http://www.dlr.de/rm/desktopdefault.aspx/tabid-5471/8991_read-16694/ (letzer Zugriff: 01.12.2010)

[DLR, 2010] MiroSurge – Telemanipulation in minimally invasive surgery http://www.dlr.de/rm/en/desktopdefault.aspx/tabid-3835/6288_read-9047/ (letzer Zugriff: 01.12.2010)

[E. Boctor et al., 2004] E. Boctor et al.: A Dual-Armed Robotic System for Intraoperative Ultrasound Guided Hepatic Ablative Therapy: A Prospective Study. Proceedings of the 2004 IEEE International Conference on Robotics & Automation

[F. Pierrot et al., 1999] F.Pierrot et al.: Hippocrate: a safe robot arm for medical applications with force Feedback. Medical Image Analysis (1999) volume 3, number 3, pp 285–300

[G.H. Glover et al., 1977] G.H.Glover et al.: Computerized Time-Of-Flight Ultrasound Tomography for Breast Examination. Ultrasound Med. Biol., Vol. 3, pp. 117-127. Pergamon Press, 1977.

[H. Gemmecke et al., 2007] H. Gemmecke et al.: 3D ultrasound computer tomography for medical imaging. Nuclear Instruments and Methods in Physics Research A 580 (2007) 1057–1065

[Intuitive Surgical, 2010] Da Vinci Surgical System http://www.intuitivesurgical.com/products/davinci_si_surgicalsystem/da-vinci-si-surgical-system-features.aspx (letzer Zugriff: 01.12.2010)

[J.Raczkowsky, 2009] Medizinische Robotik Skript 2009 J.Raczkowsky Institut für Prozessrechentechnik Automation und Robotik

[Laun, 2007] Lineare Algebra Skript 2007 Laun Medizinische Informatik Heilbronn/Heidelberg

[London South Bank University, 2010] Schalleigenschaften von Wasser http://www.lsbu.ac.uk/water/explan2.html#sound (letzer Zugriff: 01.12.2010)

[M. Janvier et al., 2008] M.Janvier et al.: Performance evaluation of a medical robotic 3D-ultrasound imaging system. Medical Image Analysis 12 (2008) 275–290

[Maier-Hein, 2010] Bildverarbeitung 2 Skript 2010 Maier-Hein Medizinische Informatik Heilbronn/Heidelberg

[Microsoft, 2005] C++/CLI Einführung http://msdn.microsoft.com/en-us/magazine/cc163681.aspx (letzer Zugriff: 01.12.2010)

[Microsoft, 2010] Interoperabilität von .NET, Win32 und COM http://www.microsoft.com/germany/msdn/library/net/SoArbeitetNETMitWin32UndCOMZusammen.mspx?mfr=true (letzer Zugriff: 01.12.2010)

[NA-MIC, 2010] Integration of OpenIGTLink into Slicer3 http://www.na-mic.org/Wiki/index.php/OpenIGTLink/Slicer (letzer Zugriff: 01.12.2010)

[NDI, 2010] Spezifikationen von Polaris Spectra und Polaris Vicra http://www.ndigital.com/medical/polarisfamily-techspecs.php (letzer Zugriff: 01.12.2010)

[O. Solberg et al., 2007] O. Solberg et al. : Freehand 3D Ultrasound Reconstruction Algorithms – A Review. Ultrasound in Med. & Biol., Vol. 33, No. 7, pp. 991–1009, 2007

[P. Arbeille et al., 2003] P. Arbeille et al.: Echographic Examination In Isolated Sites Controlle From An Expert Center Using A 2-D Echograph Guided By A Teleoperated Robotic Arm. Ultrasound in Med. & Biol., Vol. 29, No. 7, pp. 993–1000, 2003

[R.Bendl, 2010] Bildverarbeitung 2 Skript 2010 R.Bendl Medizinische Informatik Heilbronn/Heidelberg

[S. Sudha et al. , 2009] S.Sudha et al.: Speckle Noise Reduction in Ultrasound Images by Wavelet Thresholding based on Weighted Variance. International Journal of Computer Theory and Engineering, Vol. 1, No. 1, April 2009

[Siemens, 2010] Acuson S2000 ABVS http://www.medical.siemens.com/webapp/wcs/stores/servlet/ProductDisplay~q_catalogId~e_-11~a_catTree~e_100010,1007660,12761,1003803~a_langId~e_-11~a_productId~e_187041~a_storeId~e_10001.htm (letzer Zugriff: 01.12.2010)

[Siemens,2010] Siemens Somatom Definition Flash http://www.medical.siemens.com/webapp/wcs/stores/servlet/ProductDisplay~q_catalogId~e_-3~a_catTree~e_100010,1007660,12752,1008408~a_langId~e_-3~a_productId~e_187741~a_storeId~e_10001.htm (letzer Zugriff: 01.12.2010)

[T.R. Nelson et al., 1998] T. R. Nelson et al.: Three Dimensional Ultrasound Imaging. Ultrasound in Med. & Biol., Vol. 24, No. 9, pp. 1243–1270, 1998

[Techniscan Medical Systems, 2010] Techniscan SVARA WBU www.techniscanmedicalsystems.com (letzer Zugriff: 01.12.2010) und http://www.techniscanmedicalsystems.com/index.php?p=Svara%3Csup%3ETM%3C/sup%3E%20Warm%20Bath%20Ultrasound%20%28WBU%3Csup%3ETM%3C/sup%3E%29 (letzer Zugriff: 01.12.2010)

[U. Engelmann et al., 2007] U. Engelmann et al.: Das landesweite Teleradiologiekonzept für Grönland. Telemedizinführer Deutschland, Ausgabe 2007

[VDI, 1990] Montage- und Handhabungstechnik; Handhabungsfunktionen, Handhabungseinrichtungen; Begriffe, Definitionen, Symbole ; VDI