RUPRECHT-KARLS-
UNIVERSITÄT
HEIDELBERG

HOCHSCHULE HEILBRONN
TECHNIK WIRTSCHAFT INFORMATIK

Entwicklung eines Softwaretools zur Unterstützung von Registrierungsprozessen für medizinische Bilddaten auf Basis von MeVisLab

Diplomarbeit

zur Erlangung des Grads des
Dipl. -Inform. Med.
der Universität Heidelberg

Franziska Herget
Heidelberg den 19.05.2011

Referent:	Prof. Dr.-Ing. Hartmut Dickhaus
Korreferent:	Dr. med., Dipl. Phys., M. Sc. Roland Metzner
Betreuer:	Dipl.-Inform. Med. Urs Eisenmann

Zusammenfassung

Das Registrieren medizinischer Bilddatensätze ist ein komplexer und zeitintensiver Prozess. Ohne die Entwicklung effizienter und schneller Registrierungsverfahren müssten erhebliche personelle Ressourcen in das manuelle Registrieren investiert oder teilweise ganz auf deren Resultate und den einhergehenden Erkenntnissen verzichtet werden. Daher ist es besonders wichtig Neuentwicklungen in diesem Gebiet voranzutreiben und Programmstrukturen zu entwickeln, die diese neuen Verfahren einbinden, evaluieren und anschließend optimieren können.

Das Ziel dieser Arbeit war die Entwicklung eines Softwaretools, das Registrierungsprozesse von der Vorverarbeitung über die eigentliche Registrierung bis hin zur visuellen Evaluierung unterstützt. Dabei sollte die Applikation so entwickelt werden, dass sowohl Funktionalitäten als auch Benutzeroberfläche einfach erweitert oder modifiziert werden können.

Zu Beginn der Entwicklung musste ein geeignetes Framework (bzw. Entwicklungsumgebung) gefunden werden. Dieses sollte sowohl eine stabile Umgebung als auch einen möglichst großen Funktionsumfang im Bereich des Prä- und Postprocessing der Registrierung bieten können. Zudem sollte diese Entwicklungsumgebung auch Strukturen bieten, die es ermöglichen neue Funktionalitäten einfach hinzuzufügen. Auf Grund der Ergebnisse der durchgeführten Analyse kam im Rahmen dieser Diplomarbeit MeVisLab zum Einsatz.

Nach einer Aufstellung der Anforderungen an die Anwendung wurde die Konzeptionierung der Architektur und des Workflows zusammengestellt. Dabei stellte sich heraus dass der klassische Netzwerkansatz nicht in Frage kam, stattdessen wurde auf eine Netzwerkstruktur um einen zentralen Puffer gesetzt. Neben der Entwicklung der Funktionalitäten stand der Entwurf einer intuitiv bedienbaren Benutzeroberfläche im Mittelpunkt. Diese wurde dem Workflow des Registrierungsprozesses nachempfunden und mit Hilfe einiger Richtlinien optimiert.

Im Anschluss wurden die gewonnen Erfahrungen, unter Anderem mit der gewählten Entwicklungsumgebung, erörtert. Zudem wurde diskutiert ob die Anwendung die an sie gestellten Anforderungen erfüllen konnte und inwiefern der Wunsch nach einer Entwicklungsumgebung auf hohem Abstraktionsniveau die Qualität und Performanz von Neuentwicklungen beeinflussen wird.

Danksagung

Ich möchte mich an dieser Stelle bei all denen bedanken, die mich während meines Studiums und bei der Anfertigung meiner Diplomarbeit so kräftig unterstützt haben.

Meinem Freund Christian danke ich für seine Ausdauer und seine Geduld während des gesamten Studiums.

Diese Diplomarbeit möchte ich mich meinen Eltern Elke und Wolfgang widmen, die mein Studium erst ermöglicht haben, mich all die Jahre hinweg tatkräftig unterstützt haben und meinen Plänen und Wünschen gegenüber immer offen waren.

Inhaltsverzeichnis

1 Einleitung

Mit der Entwicklung der Computertomographie in den 1970er Jahren begann eine neue Etappe in der Medizin. Erstmals war es möglich dreidimensionale Aufnahmen eines Körpers aufzuzeichnen und darzustellen. Diese volldigitalen Aufnahmen ermöglichten außerdem das Modifizieren und Aufbereiten der medizinischen Bilddaten mit mathematischen statt optischen Verfahren und erschufen so ein ganz neues Wissenschaftsfeld – das der medizinischen Bildverarbeitung.

Ob als Navigationshilfe in der OP-Planung, Basis für die Strahlentherapie oder die Auswertung ganzer Patientenstudien – Computergestützte Verfahren zur Aufnahme, Bearbeitung oder Visualisierung medizinischer Daten sind nicht mehr aus der medizinischen Praxis wegzudenken.

Seit 2003 widmet sich das Institut für medizinische Biometrie und Informatik (IMBI) in Heidelberg der Entwicklung von computergestützten Methoden und Systemen für die Diagnostik, Therapeutik und Informationsverarbeitung in der Medizin. Darunter fallen auch die Entwicklung neuer bildverarbeitender Algorithmen sowie deren qualitative wie quantitative Analyse. Das Feld der Softwarelösungen in diesem Bereich ist recht groß – allerdings ist die Mehrzahl dieser Applikationen sehr komplex, aufgabenspezifisch und in sich abgeschlossen implementiert. In der Regel handelt es sich dabei oft um Applikationen die von der gleichen Firma stammen wie das bildgebende Gerät.

Für neu entwickelte Algorithmen hingegen bedarf es einer offenen Programmstruktur, die die Integration und Analyse dieser Neuentwicklungen unterstützt, ohne dass verwaltende Komponenten und Datenstrukturen von Grund auf neu implementiert werden müssen.

Solch eine Struktur stellen gleich mehrere anerkannte Programm-Bibliotheken auf dem freien Markt zur Verfügung. Sie bieten vorimplementierte Komponenten für die zweidimensionale und dreidimensionale Visualisierung, Bildverarbeitung und Evaluierung an, die nach dem eigenen Bedarf modifiziert und gekoppelt werden können. Die Ansätze dieser Bibliotheken reichen von der konservativen bis hin zur visuellen Programmierung. Die Eignung einer Bibliothek hinsichtlich einer bestimmten Fragestellung hängt von der Aufgabenstellung sowie dem zur Verfügung stehenden Zeitfenster ab. Die Bibliotheken sind entweder auf Applikationen für Endanwender, mit bereits parametrisierten Aufgaben, spezialisiert oder bieten in allen Facetten modifizierbare Endprogramme an, die dem Entwickler die Möglichkeit geben alle Komponenten hinsichtlich der Aufgabenstellung zu optimieren.

1.1 Motivation

Ein Großteil der bisher entwickelten Programme und Anwendungen aus dem Gebiet der Registrierung sind auf eine konkrete Aufgabenstellung hin spezialisiert worden. Es fehlt an einer allgemeinen Softwarebasis welche den Registrierungsablauf im Bereich der medizinischen Bildverarbeitung unterstützt und eine visuelle Evaluierung begünstigt.
Viele der bereits vorhandenen Anwendungen haben außerdem die Tatsache gemein, dass es sich bei ihnen um abgeschlossene Entwicklungen handelt, die wenn überhaupt nur bedingt Modifikationen oder Neuentwicklungen innerhalb der eigenen Struktur zulassen.

Gerade für studentische Arbeiten wäre eine offene Struktur von Vorteil, durch die man eigene Neuentwicklungen in fertige Anwendungen einfach einpflegen und testen zu können. Um die Einarbeitungszeit so gering wie möglich zu halten, müsste das Arbeiten auf einem hohen Abstraktionsniveau ermöglicht werden.

1.2 Zielsetzung

Das Ziel dieser Diplomarbeit ist es, eine Applikation zu entwickeln, die das Registrieren von medizinischen Bilddatensätzen ermöglicht und in diesem Zuge die entsprechenden vor- und nachbereitenden Verfahren unterstützt. Dazu gehört im Rahmen der Vorverarbeitung das Filtern der Bilddaten und die Segmentierung von anatomischen Strukturen. Nach der eigentlichen Registrierung sollen Möglichkeiten zur visuellen Evaluierung der Ergebnisse in 2D und 3D sowie das Sichern der Ergebnisse vorhanden sein.
Zudem sollte die zu entwickelnde Anwendung in Zukunft als Basis für studentische Projekte dienen, in denen neu entwickelte Algorithmen und Verfahren einfach einzubauen sein sollen. Die Anwendung soll also ein entsprechendes Erweiterungskonzept vorweisen. Im Vorfeld der Entwicklung muss eine Bibliothek oder ein Framework gesucht werden, welches die Anforderungen erfüllen kann. Das verwendete Toolkit soll dabei auf einer hohen Abstraktionsebene angesprochen werden können um Einarbeitungszeiten für nachfolgende Entwickler möglichst gering zu halten. Zudem sollte es eine flexible Struktur besitzen um das Erweiterungskonzept implementieren zu können.

1.3 Gliederung

In der nachfolgenden Arbeit wird beschrieben wie die Applikation MevisMatch in Hinsicht auf diese Anforderungen entwickelt wurde.
Zu Beginn steht die Analyse von frei verfügbaren Bibliotheken die diese Anforderungen an die spätere Applikation erfüllen. Nach einer praktischen Auswertung der Frameworks MITK und MeVisLab stellte sich letzteres als am geeignetsten heraus. Darauf folgt eine kurze Auseinandersetzung mit den Datenstrukturen, dem Aufbau und dem Entwickeln mit MeVisLab. Das darauffolgende Kapitel Konzeption beinhaltet eine kurze Aufstellung der Anforderungen an die Anwendung sowie die Vorstellungen der ausgewählten Architektur- und Benutzeroberflächenkonzepte. Hiernach werden die während der Entwicklungsphase implementierten Komponenten sowie die gewonnen Erfahrungen erläutert. Anschließend werden die Ergebnisse samt der Handhabung der Applikation erläutert und in Hinblick auf die Aufgabenstellung diskutiert.

2 Grundlagen

2.1 Medizinische Bildanalyse und -verarbeitung

Im Folgenden wird ein Überblick über die Akquisition und Verarbeitung von medizinischen Bilddaten gegeben. Der Schwerpunkt liegt dabei auf der Vor- und Nachbereitung von Registrierungsprozessen.

Die verschiedenen, in dieser Arbeit berücksichtigten, Bearbeitungsstufen dieses komplexen Prozesses werden in Abbildung 1 dargestellt und in den folgenden Kapiteln erläutert.

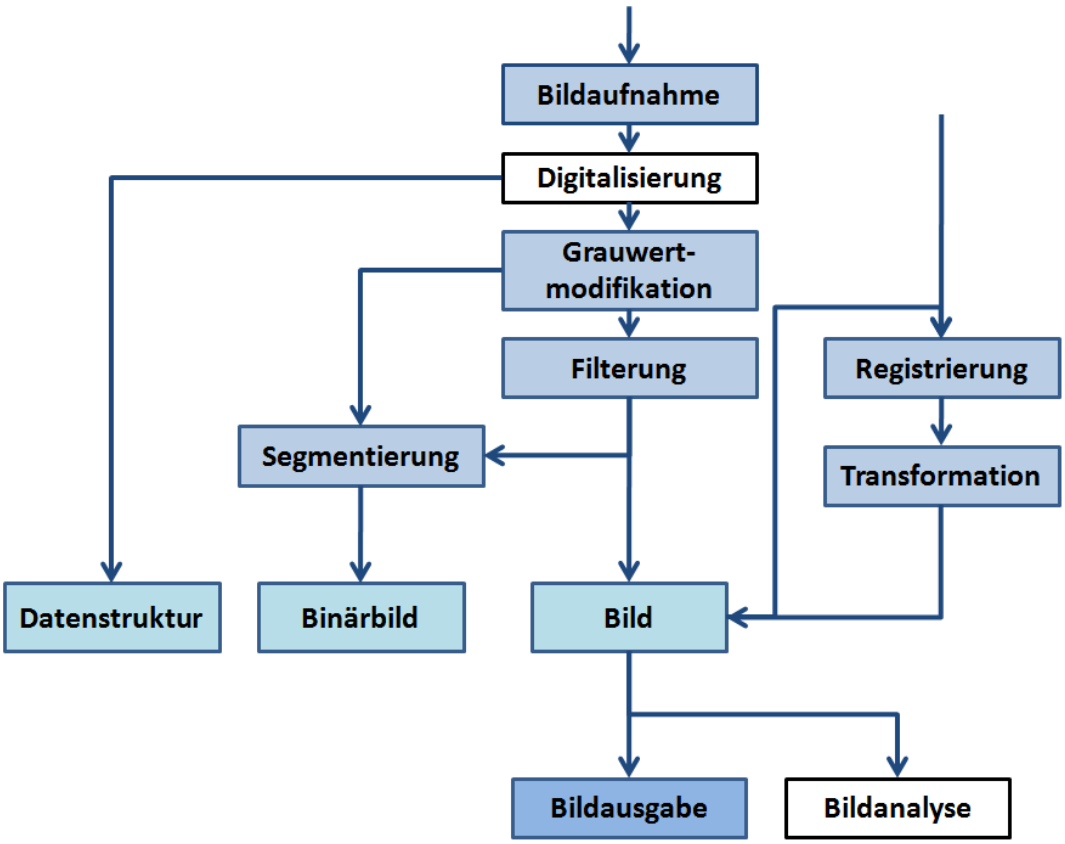


Abbildung 1: Bildverarbeitungsprozess nach [Jähne10] , blau markierte Objekte werden in den folgenden Kapiteln erläutert

Zu Beginn jeder Verarbeitung steht die Bildaufnahme mit einem entsprechenden bildgebenden Verfahren (Kapitel 2.1) und das Abspeichern des Bildes in einer geeigneten Datenstruktur (Kapitel 2.1.2). Bei den ersten Verarbeitungsschritten werden verschiedene Verfahren angewendet, die die Bilddaten für die entsprechende spätere Anwendung optimieren, dazu gehört auch das Filtern der Bilder (Kapitel 2.1.3). Für das Identifizieren und Freistellen von Objekten innerhalb der Bilder bedarf es sogenannter Segmentierungsverfahren (Kapitel 2.1.4), die für jedes Bildpixel definieren, ob es zum erkannten Objekt gehört oder nicht. Das Ergebnis dieser Segmentierung ist ein Binärbild. Zum Vergleich mehrerer Bilddatensätze ein und desselben Patienten müssen diese aufeinander ausgerichtet werden, dies erfolgt durch sogenannte Registrierungsverfahren (Kapitel 2.1.5).

2.1.1 Bildgebende Verfahren in der Medizin

Bildgebende Verfahren sind ein essentieller Bestandteil der Diagnostik und Therapieplanung in der Medizin. Bei denen im Rahmen dieser Arbeit zur Verfügung stehenden Bilddaten handelt es sich ausschließlich um Tomographien.

Das Wort Tomographie leitet sich von dem griechischen Wort „tomos“ ab und bedeutet so viel wie „Schicht“ oder „Scheibe“. Darunter versteht man das schichtweise Abbilden eines Objekts mit Hilfe eines mit Wellen bildgebenden Verfahrens.
Für jede Objektschicht werden dabei aus unterschiedlichen Richtungen mehrere Aufnahmen erstellt. Aus den Projektionen dieser Aufnahmen auf das Träger- oder Empfangsmedium (Detektor) wird mit Hilfe eines Computers die interne Struktur dieser Schicht berechnet. Ein Schnittbild gibt die inneren Strukturen so wieder, wie sie beim Durchschneiden des Objekts vorliegen würden.

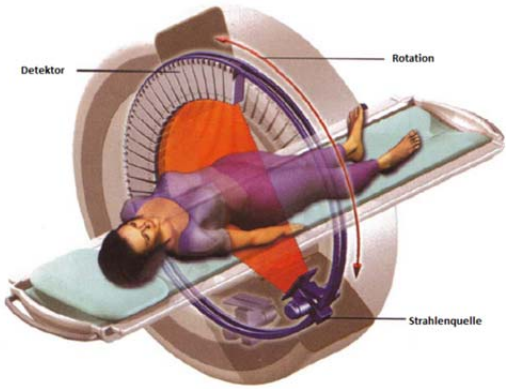


Abbildung 3: Beispiel eines Tomographen aus [Preim07]

Abbildung 2: Bildartefakt verursacht durch eine Zahnplombe

Im Gegensatz zu einem zweidimensionalen Aufnahmeverfahren wie bei der Röntgenverfahren, werden dabei keine Strukturen von anderen überlagert. Aus mehreren Schichtbildern (**Serien**) lässt sich anschließend ein dreidimensionales Volumenbild errechnen. Jeder Bildpunkt (Pixel) im Schnittbild entspricht einem volumetrischen Bildpunkt (Voxel, engl. volumetric pixel) wobei die Höhe des **Voxel**s der Schichtdicke der Aufnahme entspricht.

Die unterschiedlichen Verfahren zeichnen entweder morphologische oder funktionelle Strukturen auf.

2.1.2 Bilddaten und Datenstrukturen

Wie im vorherigen Kapitel beschrieben handelt es sich bei den Bildaufnahmen um zweidimensionale Schichtbilder. Im Ausgangszustand ist also nur eine axiale (Richtung der Körperachse) Sicht auf die Bilder vorhanden.
Neben der axialen (auch **transversalen**) Standardorientierung spricht man auch von der **frontalen** oder der **saggitalen** (seitlich) Sicht auf das Aufnahmeobjekt. Diese beiden zusätzlichen Standardorientierungen werden mit Hilfe der **Multiplanaren Reformation** (MPR) generiert. Die MPR erlaubt es aus den transversalen Schnitten beliebige neue Schnitte zu berechnen, inklusive frontalen, saggitalen, schrägen oder kurvenförmigen Schnitten.

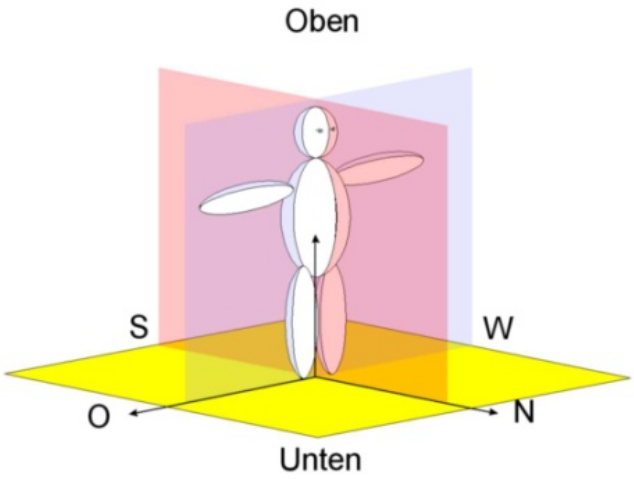


Abbildung 4: Rot Saggitalebene, Gelb Transversaleben, Blau Frontalebene

Abbildung 5: links.
Standardsichten auf eine Schädel-CTA.
RO - Transversal, LU - Saggital , RU - Frontal
Screenshot entnommen aus MeVisLab

Die Bilddaten besitzen als Minimum eine dreidimensionale räumliche Ausdehnung in *x-*, *y-* und *z-*Richtung. Zusätzlich können sie noch eine Zeitdimension *t* (für engl. *time*) besitzen. Die Zeitdimension wird für Aufnahmen über einen gewissen Zeitraum benötigt, wie es bei MRT-Aufnahmen vom schlagenden Herzen der Fall ist.

Der etablierte Standard für die Verwaltung, Speichern und Austausch von medizinischen Bildern und damit verbundener Informationen ist der „Digital Imaging and Communications in Medicine“ (DICOM) Standard. Der sogenannte DICOM-Header beinhaltet mehrere Metadaten zum Bilddatensatz wie etwa die Patienten ID, die aufnehmende Modalität, die Schichtdicke bzw. Reihenfolge oder die Serien ID. DICOM wird auch zur Bildarchivierung in sogenannten „Picture-Archiving-and-Communication-System“ (PACS) Systemen verwendet.

2.1.3 Filterung

Die Aufgabe eines Filters ist die Verbesserung der sichtbaren Bildinformationen. Das beinhaltet die Korrektur aufnahmebedingter Fehler, das Hervorheben informativer Bildinhalte oder die Restaurierung einer Textur oder Musters [Handels09] . Dabei gilt es immer eine maximale Milderung der Bildstörung bei minimalem Informationsverlust zu erhalten [Lehmann97] . Die Filterung wird durch Anwendung einer mathematischen Funktion oder Abbildung (Filter) auf das Ausgangsbild erreicht.

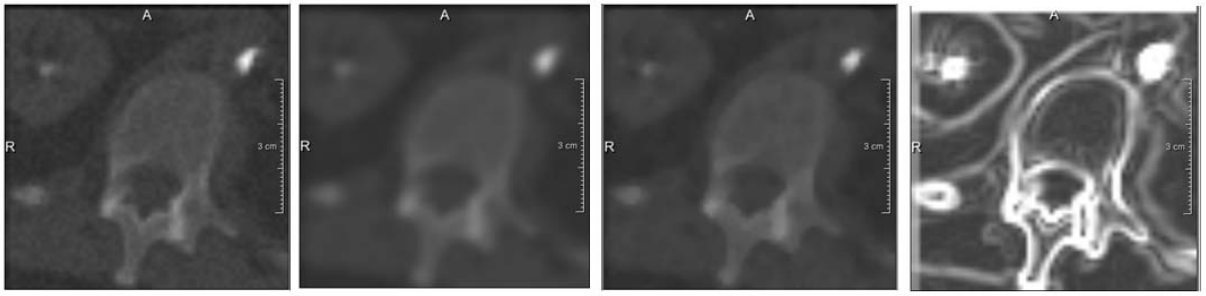


Abbildung 6: Filterungen eines künstlich verrauschten Bildes
e.v.l. künstlich verrauschtes Ausgangsbild, z.v.l. Gauß-Filter, z.v.r. Medianfilter, e.v.r. Sobel Operator

Die Filter werden in **Punktoperationen** (neuer Wert abhängig vom aktuellen Wert), **lokale Operationen** (abhängig von Nachbarpixeln) und **globale Operationen** (abhängig vom gesamten Bildinhalt) aufgeteilt, näheres kann in [Jähne10] und [Lehmann97] nachgelesen werden. Im Rahmen dieser Arbeit wurden die Filter **Rangordnungsfilter** (Median, Min- und Maxfilter), **Gaußfilter**, **Sobel Operator** und **morphologische Filter** (Erosion und Dilatation) verwendet. Diese werden nun im Folgenden erläutert.

Rangordnungsfilter

Beim diesem Filtertyp handelt es sich um einen lokalen Filter, bei dem aus der angewandten Matrix die Werte extrahiert, sortiert und der neue Wert anhand einer Regel ausgesucht wird. Dies wird in Abbildung 7 am Beispiel vom **Median** dargestellt. Der Median einer Gruppe von Werten wird definiert als der mittlere Wert in einer geordneten Auflistung dieser Variablen [Birk11] .

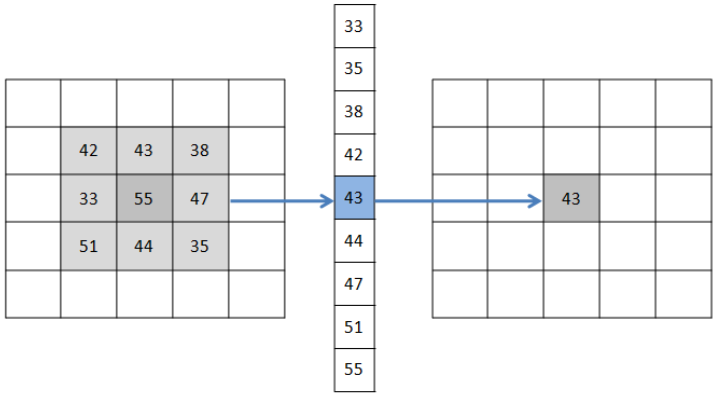


Abbildung 7: Beim Median Filter werden die Werte aus der Matrix geordnet und der mittlere Wert in das Ergebnisbild geschrieben.

Der Median-Filter eignet sich sehr gut zum Entfernen von binärem Rauschen (schwarze und weiße Pixel) ohne dass dabei im Bild vorhandene Kanten verschwimmen (s. Abbildung 6) Analog zum Median gibt es noch die Rangordnungsfilter für den **minimalen** und den **maximalen** Intensitätswert.

Gauß-Filter

In der Bildverarbeitung wird der Gauß-Filter zur Glättung oder Weichzeichnung der Bildinformationen verwendet (s. Abbildung 6). Er eignet sich sehr gut für ein gleichverteiltes Rauschen, kleine Strukturen verschwinden während große Strukturen erhalten bleiben. Ein Nachteil hingegen ist, dass Kanten oft verschwimmen.

Sobel Operator

Der Sobel Operator wird zum Betonen und Hervorheben von Kanten angewandt. Kanten können am einfachsten identifiziert werden, indem Zeile für Zeile die Ableitung des Bildes berechnet wird. Hohe bzw. niedrige Werte in dieser Ableitung stehen für starke Steigungen im Ausgangsbild, also für Kanten (s. Abbildung 6, rechts.). Der Sobel Operator arbeitet zeilenweise, je nach gewählter Bildzeile und Richtung der Ableitung, werden unterschiedliche Kanten betont.
Im Anschluss an die Ableitung wird das Ergebnisbild mit Hilfe eines Gauss-Filters geglättet [Birk11] .

Morphologische Filter

Unter diesem Filter werden die **Erosion** (verringern) und **Dilatation** (ausdehnen) zusammengefasst, sie werden in der Regel auf Binärbilder angewandt um die segmentierten Strukturen (s. Kapitel 2.1.4) besser voneinander trennen zu können.
Mit der Erosion werden Objekte verkleinert und schmale Verbindungen zwischen Objekten getrennt. Dem Pixel im Ergebnisbild wird dabei der Minimalwert der angewandten Matrix zugewiesen. Dies ist in Abbildung 8 dargestellt.

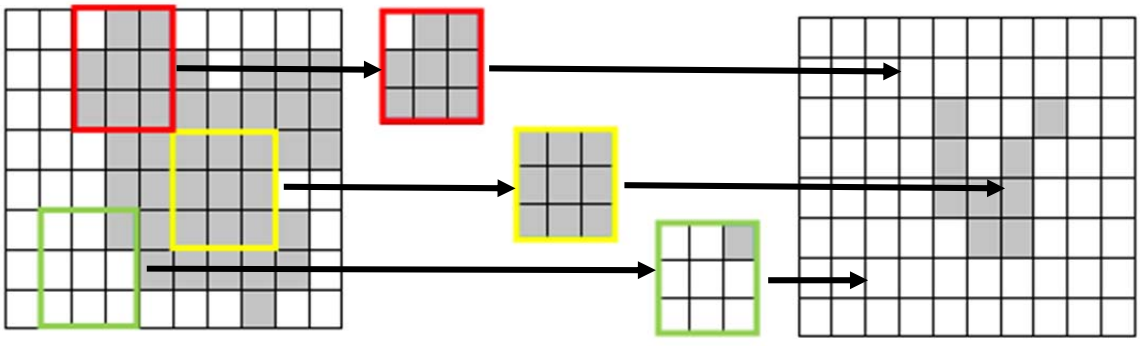


Abbildung 8: Erosion eines Bildes. Links: Ausgangsbild, Rechts: Ergebnisbild

Ebenfalls Bestandteile dieser Filtergruppe sind das **Closing** (engl. für Schließen) und das **Opening** (engl. für Öffnen).
Beim **Closing** handelt es sich um eine Ausführung der Dilatation mit anschließender Erosion. Dabei werden Lücken zwischen einzelnen Objekten geschlossen. Beim **Opening** hingegen werden schwache Verbindungen zwischen Strukturen getrennt, durch eine Erosion mit anschließender Dilatation.

2.1.4 Segmentierung

Das Ziel der Segmentierung ist medizinisch oder diagnostisch relevante Strukturen, wie etwa Tumorgewebe, Nervenbahnen oder Blutgefäße, von anderem Gewebe oder Objekten zu separieren [Handels09] . Im Kontext dieser Arbeit wird das Ergebnis der Segmentierung entweder als Basis für die Registrierung verwendet (s. Kapitel 2.1.5) oder zur Beurteilung der Lage der verschiedenen Strukturen im Raum zueinander.
Die Segmentierungsverfahren werden in vier verschiedene Methodengruppen eingeteilt: **pixelbasiert**, **kantenbasiert**, **flächenbasiert** und **modellbasiert,** diese werden in [Jähne10] näher erläutert. Alle Verfahren haben dabei das gleiche Ziel: die Zusammenfassung von benachbarten Voxeln zu zusammenhängenden Regionen. Das Ergebnis jeder Segmentierung ist ein sogenanntes **Binärbild**, die segmentierte Struktur wird auf Voxelebene binär repräsentiert (1 für Objekt, 0 für Hintergrund). Im Fall von Abbildung 9 links unten, wird die segmentierte Struktur in Weiß, der Rest in Schwarz dargestellt.

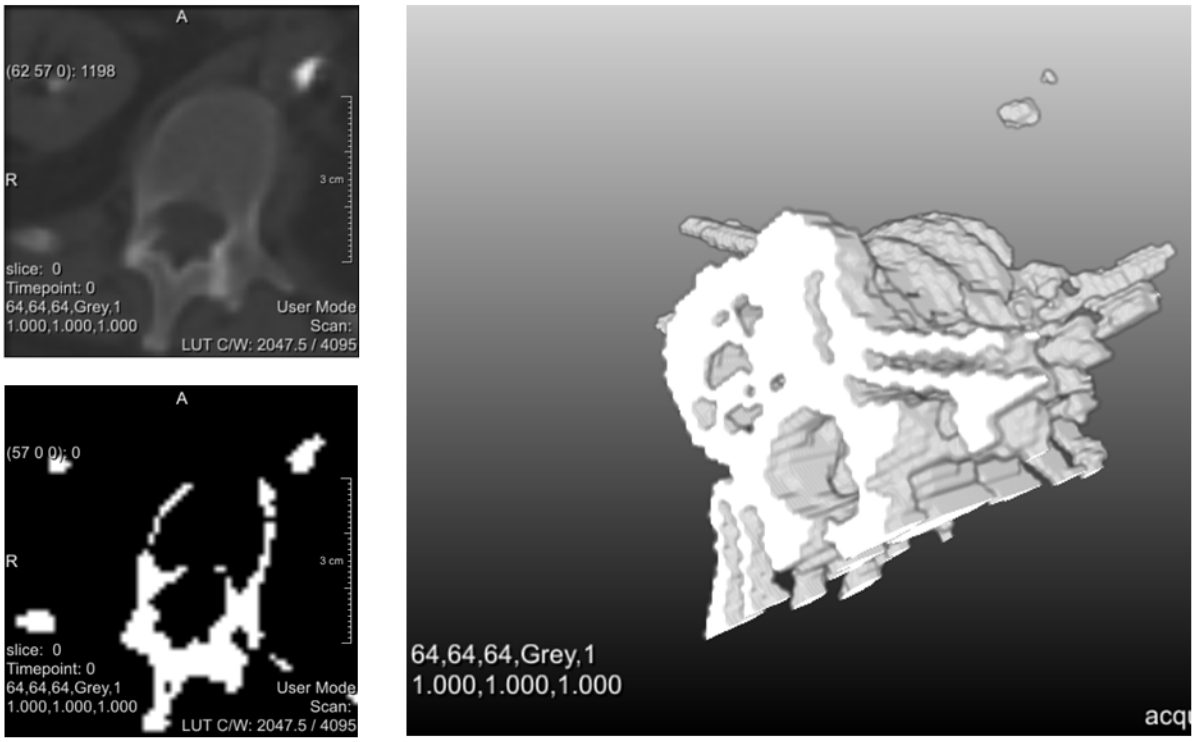


Abbildung 9: **Segmentierungsergebnis nach einem einfachen Thresholdverfahren.**
LO - Ausgangsbild, LU - Binärbild, R - Dreidimensionale Ansicht der segmentierten Struktur

Zu den in dieser Arbeit verwendeten Segmentierungsverfahren gehören das **Threshold-,** das **Fuzzy Cluster**- und das **Region-Growing-Verfahren**.

Threshold

Der Threshold ist ein pixelbasiertes Verfahren, bei dem Voxel der segmentierten Struktur zugeordnet werden, je nachdem ob ihre Intensität über -oder unterhalb eines bestimmten Schwellenwertes liegt. Die Segmentierung in Abbildung 9 wurde mit Hilfe eines solchen Threshold-Verfahrens durchgeführt. Dieses Verfahren eignet sich besonders zur schnellen Segmentierung von farblich stark hervorgehobenen Strukturen wie etwa Knochen in einer CT-Aufnahme.

Region-Growing

Dieses Verfahren ist ein regionenbasiertes Segmentierungsverfahren. Wie oben erläutert werden vom Anwender gezielt Markierungen in das Ausgangsbild gesetzt und ein Ähnlichkeitsmaß definiert. Anschließend wird das Bild ausgehend von diesen Pixeln segmentiert (Abbildung 10).

Diese Segmentierungstechnik macht sich die gleiche mathematische Eigenschaft zu Nutze wie der bereits erläuterte Sobel Operator, dass die Position einer Kante durch einen hohen Wert in der ersten Ableitung definiert ist. Diese Kante wird in einer zweiten Ableitung als Nulldurchgang angezeigt.
Das Verfahren sucht in der Nachbarschaft des Ausgangspixels nach weiteren Nulldurchgängen. Diese werden anschließend miteinander verbunden. Die Segmentierung ist als abgeschlossen anzusehen, wenn der Algorithmus das Objekt anhand seiner Nulldurchgänge umrundet und wieder am Ausgangsbildpunkt angekommen ist.

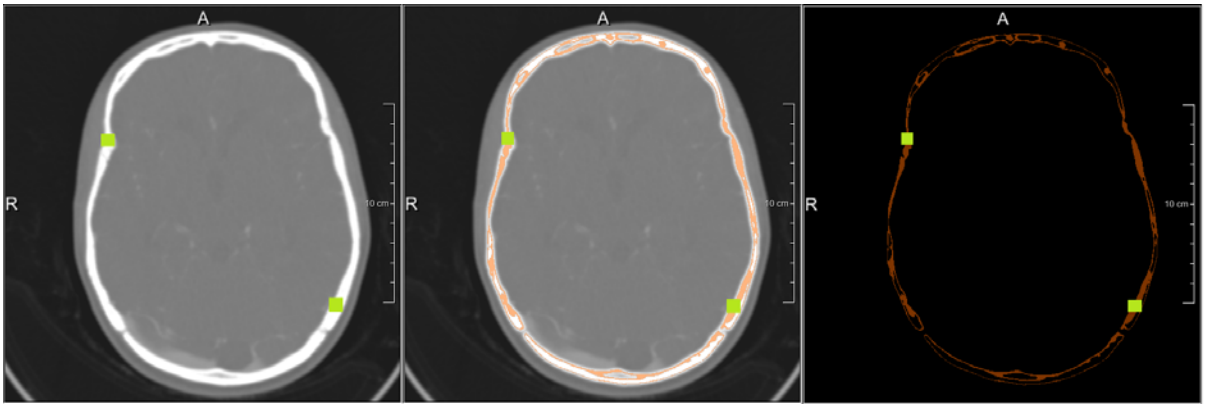


Abbildung 10: **Region-Growing Verfahren.**
Links: Marker auf Knochenstruktur
Mitte: Segmentierte knöcherne Strukturen mit Bilddatensatz
Rechts: Segmentierte knöcherne Strukturen ohne Bilddatensatz

Fuzzy-Cluster

Beim Fuzzy-Clustering wird ein Voxel nicht einer Struktur zugeteilt, sondern erhält eine prozentuale Wahrscheinlichkeit mit der es zu dieser Struktur gehören könnte. Ein Voxel kann dabei mehreren Objektgruppen (Cluster) zugesprochen werden, daher die Bezeichnung „Fuzzy“ für ausgefranst oder unscharf. Im Gegensatz zur allgemeinen Segmentierung werden dabei mehrere Binärbilder ausgegeben, je nach Anzahl der Cluster. Der FuzzyCluster-Algorithmus baut auf dem Fuzzy C Means Algorithmus zum Bilden von Clustern auf.

2.1.5 Registrierung

Die Aufgabe der Registrierung ist es, mehrere medizinische Bilddatensätze so aufeinander auszurichten, dass sie mit größtmöglicher Übereinstimmung der anatomischen Strukturen übereinander gelegt werden können. Dies ermöglicht eine vergleichende Darstellung der verschiedenen Datensätze. Aus den kombinierten Daten lassen sich neue und detailliertere Informationen gewinnen.
Methodisch bedeutet dies, dass die Koordinatensysteme der unterschiedlichen Bilder angepasst und in ein gemeinsames transformiert werden. Es wird dabei in einem Registrierungsdurchlauf jeweils nur ein Bild auf ein anderes Bild oder Koordinatensystem registriert. Der Bilddatensatz auf dem ausgerichtet wird bezeichnet man als **Referenzbild** oder „Fixed Image“ (engl. für unbewegliches Bild). Dem gegenüber steht das **Objektbild** oder „Moving Image“ (engl. für bewegliches Bild). Das Objektbild wird dabei auf das Koordinatensystem des Referenzbildes transformiert. Im Falle von klinischen Studien wird oft nur auf ein normiertes Koordinatensystem registriert. Das Ergebnis dieses Verfahrens wird durch eine Transformationsmatrix dargestellt.

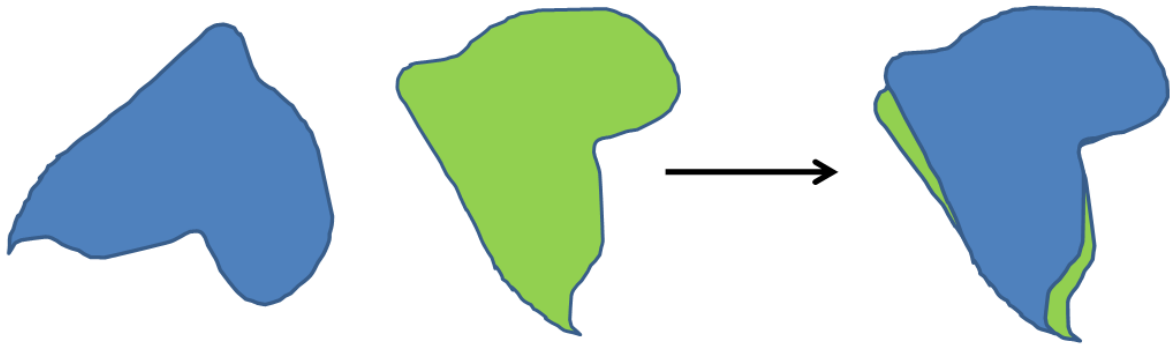


Abbildung 11:Transformation des Objektbilds (blau) zu einer annähernd deckungsgleichen Position zum Referenzobjekt (grün)

Im Zuge der bildgestützten Diagnostik und Therapieverfahren nimmt das Registrieren oder Matchen einen hohen Stellenwert ein. So wird die Bildregistrierung erfolgreich eingesetzt zur Fusion von Bildern unterschiedlicher Aufnahmezeitpunkte, unterschiedlicher Volumina oder unterschiedlichen Modalitäten.
Es ist nicht möglich zweimal von einem Patienten eine zu den Koordinaten des Aufnahmegerätes identische Bildaufnahme zu tätigen. Minimale Körperbewegungen, wie sie allein schon bei der Atmung vorkommen, bewirken eine Verformung oder Verschiebung des Aufnahmebereichs. Bei Aufnahmen zu späteren Zeitpunkten kann die frühere Lage des Patienten zum aufnehmenden Gerät nie vollständig nachgestellt werden. Dies können auch Fixierungstechniken [Schlegl07] , wie sie zum Beispiel in der Strahlentherapie angewendet werden, nicht vollständig kompensieren.

Die Registrierung von multimodalen Aufnahmen ist besonders für die Deutung funktionaler Daten, wie sie beim PET oder SPECT vorkommen von essentieller Bedeutung. Diese funktionalen Daten können erst durch das Registrieren mit einem CT oder MRT den eigentlichen Körperregionen zugeordnet werden.

Bei den Registrierungsalgorithmen unterscheidet man drei Ansätze, den **bildbasierten** Verfahren (auf Basis der Bildinformationen), den **landmarkenbasierten** Verfahren (auf Basis manuell oder automatisch gesetzten Markierungen) oder den **geometriebasierten** Verfahren (auf Basis segmentierter Strukturen oder geometrischen Modellen) [Handels09] .

Bei den **bildbasierten** Verfahren sind die Grau- oder Intensitätswerte der einzelnen Pixel die Basis auf der registriert wird. Dabei wird das Objektbild solange über dem Referenzbild verschoben bis die Differenzen zwischen den Intensitätswerten der Pixel minimal werden.

Das **landmarkenbasierte** Registrieren funktioniert ohne direkten Bezug zu den Bildinformationen. Ausgangsdaten sind manuell festgelegte Punkte in beiden Bilddatensätzen, die sogenannten Landmarken. Diese werden vom Anwender auf markante Bildpunkte gesetzt und können später zur Navigation oder einer späteren quantitativen Analyse verwendet werden.

Vor der Registrierung werden möglichst kleine, anatomisch eindeutige Strukturen markiert, deren relative Position zu den anderen Bildinhalten konstant bleibt. Diese Strukturen müssen in beiden Bildserien erkennbar sein. Dabei kann es sich um morphologische Strukturen handeln oder um vorher implantierte oder aufgesetzte Marker. Für eine landmarkenbasierte Registrierung sind mindestens drei korrespondierende Punktpaare notwendig. Diese dürfen hat auf einer Gerade liegen, da nur drei voneinander linear unabhängige Vektoren eine Ebene beschreiben [Wiki3] . Bei den landmarkenbasierten Verfahren gibt es zwei unterschiedliche Ansätze, die entweder auf **Punktpaaren** oder **Punktwolken** basieren. Erstere versuchen den durchschnittlichen Abstand zwischen zwei Punkten in Referenz- und Objektbild zu verringern, letztere die beiden Punktwolken möglichst deckungsgleich übereinander zu legen.

Die **geometriebasierten** Registrierungsverfahren gehen von in beiden Bildern festgelegten Formen aus. Dies können Oberflächen, Linien oder einfache geometrische Figuren wie Kugeln, Zylinder etc. sein. Für die weiten Erläuterungen werden diese auf die segmentierten Strukturen beschränkt. Diese segmentierte Struktur des Objektbildes wird mittels Skalierung, Rotation und Deformierung auf die andere angepasst.

Beispiele für diese Registrierungsverfahren werden in Kapitel 2.1.5.1 beschrieben.

Bei Registrierungsbasen wird ferner zwischen **extrinsischen** (körperfremden) und **intrinsischen** (körpereigenen) Merkmalen unterschieden. Nach [Suri05] basieren extrinsische Verfahren auf Objekten die dem Patienten aufgeklebt oder implantiert werden.

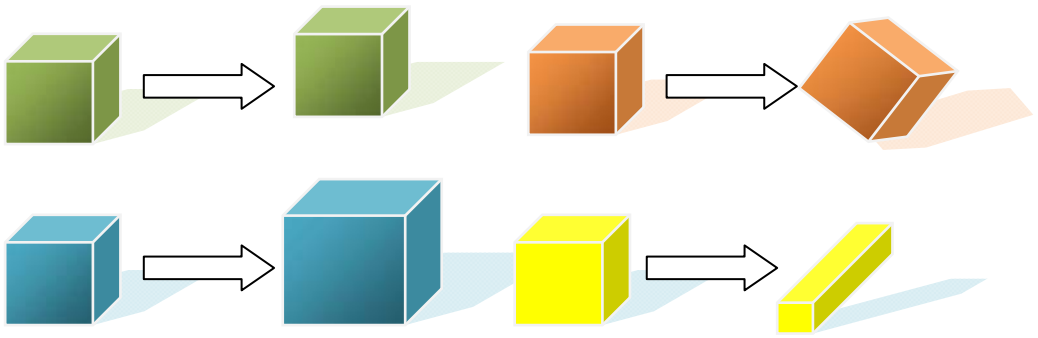


Abbildung 12: Darstellung der verschiedenen Transformationen.
LO - Translation RO – Rotation
LU - Skalierung RU - elastische Deformation

Die Ergebnistransformationen werden in **rigide**, **affine** und **elastische** Transformationen unterteilt [Suri05] , die einzelnen Transformationsverfahren werden in Abbildung 12 dargestellt.

Die **rigide** Transformation ändert nichts an der Struktur oder der Form des Objekts. Sie erlaubt das Verschieben (Translation), die Rotation und die Spiegelung des Objekts. Dabei bleiben die räumlichen Zusammenhänge innerhalb des Objekts erhalten.
Diese Transformation eignet sich besonders für Objekte, die eine feste Struktur besitzen und sich in den Aufnahmen nur in ihrer Position und Ausrichtung unterscheiden.

Bei der **affinen** Transformation ist bezeichnend dass vorher parallele Strukturen nach der Transformation ebenfalls parallel zueinander sein. Im Vergleich zu den Funktionen der rigiden Transformation kommen hier noch die Skalierung (vergrößern oder verkleinern) und die Scherung (auftrennen oder einschneiden, sodass das Ausgangsbild auseinanderklafft) hinzu.

Bei der **elastischen** Transformation werden das Objektbild oder Auszüge davon (z.B. einzelne Schichten) elastisch deformiert, um sie an das Referenzbild anzupassen. Diese Transformation wird zumeist bei 2D zu 3D bzw. 3D zu 3D Registrierungen verwendet [Suri05] .

2.1.5.1 Registrierungsverfahren und ihre Anwendung

Für jeden Anwendungsfall eignet sich ein anderes Registrierungsverfahren. In der Praxis haben sich verschiedenste Verfahren etabliert, die alle auf eine bestimmte Aufgabenstellung hin optimiert werden.

Nach dem **Handbook of Biomedical Image Analysis** [Suri05] werden Registrierungsverfahren nach sieben Eigenschaften eingeordnet:

Dimension

Die Dimensionierung eines Bildes ist ein wichtiger Faktor für die Registrierung. Wie in Kapitel 2.1.2 schon erläutert werden medizinische Bilder zweidimensional und dreidimensional aufgenommen. Bei Aufnahmen über einen gewissen Zeitraum spielt die Zeitdimension eine Rolle. Die Registrierung 2D zu 2D ist in der Regel weniger komplex als die 3D zu 3D, laut [Suri05] ist hingegen die 2D/3D Registrierung am anspruchsvollsten. Die Zeitdimension muss bei Aufnahmen beachtet werden, bei denen die Bewegung innerer Organe oder Stoffwechselvorgänge betrachtet werden.

Art der Registrierungsbasis

Wie in Kapitel 2.1.5 beschrieben, basieren die Berechnungen der Registrierungsverfahren auf den Bilddaten, gesetzten Markern oder segmentierten Strukturen. Einige Registrierungsverfahren greifen auf mehr als einen Datentyp zu, um die Registrierung im Laufe der Berechnung einschätzen und weiter optimieren zu können.
Die Registrierungsbasis ist ausschlaggebend für die Genauigkeit und den Rechenaufwand für die Registrierung. Bildbasierte Verfahren sind in der Regel komplexer als landmarken- oder maskenbasierte Verfahren. Das Registrieren anhand von extrinsischen Strukturen ist relativ einfach, schnell und kann oft komplett automatisiert durchgeführt werden.

Art der Transformation

Die Art der Transformation legt fest, welche Strukturen im Bild berücksichtigt und welche berücksichtigt und welche Komponenten für den Registrierungsalgorithmus verwendet werden müssen (s. Kapitel 2.2.4). Die in der Praxis am häufigsten verwendeten Transformationen sind die rigide und die affine. Die Berechnung des Deformationsfelds der elastischen Registrierung ist erheblich zeit- und ressourcenaufwendiger als für die der Transformationsmatrix.

Interaktion

Registrierungsverfahren werden in drei unterschiedliche Interaktionsstufen eingeteilt. Das wären zuerst die (voll-)automatischen Verfahren, bei denen der Anwender lediglich die Bilddaten zur Verfügung stellt. Bei den semiautomatischen Verfahren werden vom Anwender zusätzliche Informationen wie Landmarken oder segmentierte Strukturen bereitgestellt. Bei bestimmten Verfahren müssen vorgeschlagene Transformationen vom Anwender bewertet bzw. bestätigt werden. Beim manuellen Registrieren richtet der Anwender die Bilddaten nach visuellen oder analytischen Eigenschaften per Hand aus.

Optimierungsverfahren

Das Registrieren ist ein mehrstufiger Prozess, bei dem interne Parameter anhand von Zwischenergebnissen immer wieder optimiert werden. Optimierungsverfahren beziehen ihre Parameter entweder aus den vorhandenen Informationen oder sie werden durch ein bestimmtes, von außen her definiertes, Kriterium vorgegeben. Viele Verfahren nutzen mehrere

Optimierungsstufen, bei denen in immer kleiner werdenden Schritten das Registrierungsergebnis optimiert wird.

Modalitäten

Es gibt vier verschiedene Möglichkeiten Modalitäten zu kombinieren. Die **monomodalen** Verfahren arbeiten mit Aufnahmen vom gleichen bildgebenden Gerät. Dem entgegen stehen die **multimodalen** Verfahren, bei denen Aufnahmen unterschiedlicher bildgebender Verfahren registriert werden. Die letzten beiden Verfahren registrieren ein Bild zu einem Modell oder ein Modell zu einem Bild. Diese Verfahren werden oft für intraoperative Registrierungsaufgaben verwendet, bei denen die Lage vom Körper zu den therapeutischen Geräten berechnet werden muss.

Bezug

Es werden nicht immer Aufnahmen ein und desselben Patienten registriert. Für Studien und klinische Forschung werden außerdem Bilddatensätzen unterschiedlicher Patienten registriert, ferner werden auch Bilder eines Patienten zu einem, aus einer Bilddatenbank konstruierten, künstlichen Bild registriert.

Sowohl die Entwicklung als auch die Auswahl eines Registrierungsalgorithmus ist unter Berücksichtigung dieser sieben Eigenschaften denkbar komplex. Für jede Aufgabenstellung eignen sich unterschiedliche Verfahren.
Zu denen im Rahmen dieser Arbeit verwendeten Registrierungsverfahren gehören die **Mutual Information** und die **Closed-Form-Solution**.

Mutual Information Verfahren

Hierbei handelt es sich um ein bildbasiertes Verfahren, bei dem ein Ähnlichkeitsmaß zwischen den Bildern – die **Mutual Information** – maximiert werden soll [Neff05] . Dabei gelten zwei Bilder als ähnlich, wenn mit großer Wahrscheinlichkeit vom Intensitätswert eines Pixels auf den des koordinatengleichen Pixels geschlossen werden kann.
Laut [Wels96] sind die Mutual Information Verfahren für die multimodale Registrierung sehr gut geeignet und liefern reproduzierbare und robuste Ergebnisse [Neff05] .

Closed-Form-Solution

Dieses Verfahren ist landmarkenbasiert und arbeitet auf Punktwolken. Für die affine Registrierung wird jene Transformation gesucht, bei denen der durchschnittliche Abstand zwischen den Landmarken minimal ist, siehe [Suri05] Kapitel 1.4. Das Ergebnis dieser Registrierung ist eine affine Transformation. Dieses Verfahren arbeitet unabhängig von den Bildinformationen und ist daher auch für multimodale Registrierung geeignet.

2.1.5.2 Transformationsraum

Bei der Transformation werden zwei unterschiedliche Koordinatensysteme unterschieden, das **Bildkoordinatensystem** und dem **Weltkoordinatensystem**, diese beiden Systemen werden in den Abbildung 13 dargestellt.

Abbildung 13: Links: Bilddaten im eigenen Koordinatensystem, Rechts: Bilddaten im Weltkoordinatensystem

Wenn von der Transformation eines Bildes geredet wird, ist damit das Transformieren im Weltkoordinatensystem gemeint. Die Bilddaten werden dabei nicht manipuliert. Dies hat zur Folge, dass in normalen zweidimensionalen Viewern die Auswirkungen der Transformation zunächst nicht zu erkennen sind. Erst in Viewern mit Bezug zur Lage des Bildes im Weltkoordinatensystem, wie etwa dreidimensionale Viewer oder solche mit Overlay-Funktion, wird eine Transformation sichtbar.

2.2 Analysierte Bibliotheken zur Visualisierung und Registrierung

Unter den verschiedenen Bibliotheken für die Registrierung und Visualisierung der Bilddatensätze gibt es verschiedene Ansätze. Von proprietären Systemen von Medizintechnikherstellern bis hin zu Open-Source Projekten die von einer Vielzahl an Anwendern immer weiter entwickelt werden. Bibliotheken die unter diese Kategorie fallen, erfüllen zwar nicht alle die Restriktionen für das Medizinproduktegesetz [MPG94] und dürfen daher nicht zu diagnostischen und therapeutischen Zwecken verwendet werden, sind aber für Prototyping u.A. für studentische Arbeiten gut geeignet. Sie bieten zumeist gut dokumentierte Schnittstellen und eine Projektstruktur die Neuentwicklungen von bildverarbeitenden Algorithmen und Verfahren begünstigt. Die damit verbundene Vielzahl an Ansprechpartnern und offiziellen und privaten Dokumentationen erlauben ein schnelles Einarbeiten.

Im Folgenden werden die für diese Arbeit potentiell in Frage kommenden Bibliotheken vorgestellt. Die im Rahmen der Sichtung bereits aufgekommenen Ausschlusskriterien werden im entsprechenden Kapitel aufgeführt und erläutert.

2.2.1 Visualization Toolkit (VTK)

Bei **VTK** handelt es sich um eine Open-Source C++ Visualisierungs-Bibliothek, die besonders im wissenschaftlichen Sektor ihre Verwendung findet. Bekannt ist sie für Visualisierungen im 3D-Bereich. Dabei liegt ihr Fokus nicht nur im medizinischen Bereich, sondern wird in allen wissenschaftlichen Segmenten verwendet in denen Aufnahmen analysiert und visualisiert werden müssen.
VTK erlaubt das Entwickeln auf einem sehr niedrigen Abstraktionsniveau. Dies heißt dass die Bibliothek dem Entwickler zwar viele Möglichkeiten und Freiheiten beim Programmieren ermöglicht, erhöht aber auch die Komplexität der Implementierung. VTK wird von vielen Toolkits für die Visualisierung verwendet.

2.2.2 Open Inventor

Open Inventor wurde von Silicon Graphics International entwickelt und ist eine freie C++ basierte Bibliothek zum Erstellen von 3D-Grafiken [Ope10] . Verschiedene geometrische Objekte wie Kegel, Würfel, etc. werden in sogenannten Scene-Graphen dargestellt.
Nach [Heise2] wurde der Quellcode dieser Bibliothek 2002 als Open-Source veröffentlicht. Trotz ihres Alters (veröffentlicht 1988/89) ist sie im medizinischen, technischen oder wissenschaftlichem Sektor weiterhin verbreitet und ihre Struktur diente als Standard zur Entwicklung von Virtual Reality Applikationen [Wiki2] .

2.2.3 VolView

VolView ist ein kostenloses Volumenvisualisierungssystem der Firma Kitware für medizinische oder wissenschaftliche Datensätze [Vol10] . Die erste Version wurde 1999 entwickelt, seit November 2010 stellt Kitware das System kostenlos zur Verfügung. Den kompletten Quellcode auf Open-Source-Basis zur Verfügung zu stellen ist zwar geplant [Heise1] , aber noch nicht umgesetzt. Die Firma ist bekannt für ihre Open-Source-Produkte und bietet entsprechende kostenpflichtige Lehrgänge und Online-

Tutorials an. Obwohl mehrere Versionen von VolView für verschiedene medizinische Bereiche existieren (Zahnmedizin, Telemedizin, etc.) ist es für Außenstehende nicht möglich den Aufbau der Applikation zu modifizieren. Es ist lediglich möglich neue Funktionalitäten über ein PlugIn-Prinzip hinzuzufügen. Auf Grund dieser fehlenden Flexibilität kam dieses System nicht zur Verwendung.

2.2.4 Insight Segmentation and Registration Toolkit (ITK)

Die Bibliothek **ITK** ist eine Open-Source-C++-Klassenbibliothek für die Segmentierung, Registrierung und Analyse von medizinischen Bilddatensätzen [ITK] . Sie bietet sowohl eine Vielzahl an etablierten und fundamentalen Algorithmen als auch eine Struktur die es erlaubt neue Algorithmen modular zu entwickeln.

Der Schwerpunkt dieser Arbeit liegt auf der Registrierung, daher wird im Folgenden der Aufbau der ITK-internen Registrierungsalgorithmen erläutert wie er auch von vielen anderen Bibliotheken übernommen wird. Die Struktur eines solchen Algorithmus ist in Abbildung 14 dargestellt.

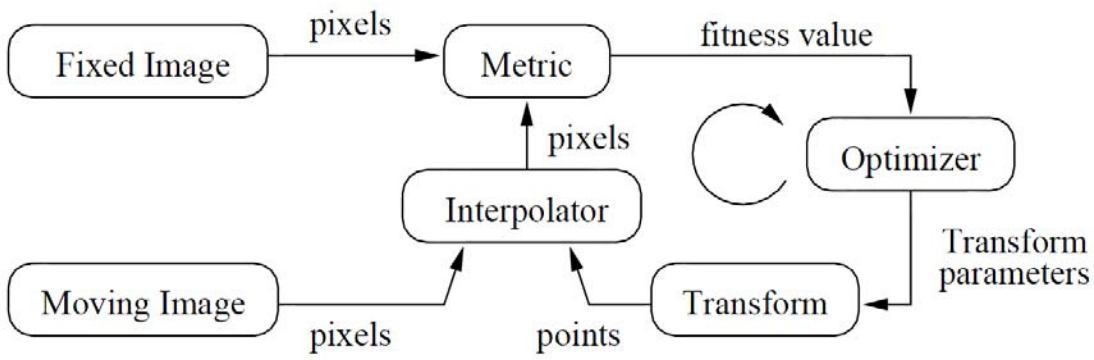


Abbildung 14: Komponenten eines Registrierungsalgorithmus entwickelt mit ITK [Ibanez03]

Ein typischer Registrierungsalgorithmus besteht laut [Ibanez03] aus einem **Interpolator**, einer **Metrik**, einem **Optimierer** und einer **Transformationskomponente**.
Nach einer Transformation liegen die einzelnen Voxel nicht unbedingt genau auf dem Voxelsystem des Referenzbildes. Daher generiert der Interpolator die Intensitätswerte für das Objektbild im Bezug auf das Voxelsystem des Referenzbildes.
Die Metrik berechnet das Ähnlichkeitsmaß der beiden Bilder nach der Registrierung und Transformation. Erreicht dieses Maß nicht einen gewissen vorgegebenen Schwellwert interpretiert der Optimierer die Daten bisheriger Transformationen aus und schlägt neue Parameter für die Transformationskomponente vor. Das Transformationsmodul führt die Transformation des Bildes aus. Wenn es möglich ist eine Rücktransformation durchzuführen, übernimmt diese Komponente ebenfalls diese Aufgabe. Jede dieser Komponenten liegt unterschiedlichen Konfigurationen vor und können beliebig gekoppelt und parametrisiert werden.

Auf die Verwendung von ITK und VTK als direkte Ausgangsbasis wurde auf Grund des niedrigen Abstraktionsniveaus verzichtet.

2.2.5 MatchPoint

Diese Open-Source Bibliothek [Floca09] wurde im Deutschen Krebsforschungszentrum Heidelberg (DKFZ) von der Arbeitsgruppe „Software Development for Integrated Diagnostic and Therapy“ (SIDT) entwickelt. **MatchPoint** basiert auf ITK und übernimmt dessen Klassenstruktur. Sie kapselt und vereinfacht ferner die Funktionen der ITK-Registrierungs-Klassen. Der große Vorteil von MatchPoint ist, dass miteinander nicht kompatible Komponenten eines Registrierungskomplexes bereits zum Zeitpunkt des Kompilierens erkannt werden. Die Bibliothek ermöglicht so ein schnelleres und effektiveres Arbeiten und Entwickeln von entsprechenden Algorithmen.
Der Aufbau eines Registrierungsalgorithmus entspricht dem von ITK.

2.2.5.1 MeVisLab-MatchPoint-Schnittstelle (Bridge)

Innerhalb der Projektgruppe SIDT zur Bibliothek MatchPoint wurde eine Schnittstelle zwischen MeVisLab und MatchPoint entwickelt [Prüm11] . Diese erlaubt das Verwenden von in MatchPoint entwickelten Registrierungsverfahren in MeVisLab. Bei der Bridge handelt es sich um ein mit C++ entwickeltes MeVisLab-Modul, welches es dem Anwender ermöglicht bereits als DLL generierte MatchPoint-Verfahren zur Ausführung zu bringen.

2.2.6 CImg

Bei dieser Bildverarbeitungsbibliothek handelt es sich um ein Projekt außerhalb des medizinischen Schwerpunkts. Sie entstand aus einer Doktorarbeit [Tschum10] aus der „Siège et centres de recherche INRIA“ in Frankreich und wird von mehreren Personen aus dem wissenschaftlichen Bereich weiterentwickelt und gepflegt. Sie wurde vom Entwickler als Basis für verschiedene Vorlesungen und Praktika verwendet. Es ist eine sehr schlanke, einfach anzuwendende und generisch entwickelte Bibliothek. Es ist möglich durch das Installieren eines Plugins DICOM Datensätze zu verarbeiten, allerdings befindet sich diese Komponente in einem frühen Entwicklungsstadium und sie wird vom Entwickler selbst als instabil bezeichnet. Daher wurde auf das Verwenden dieser Bibliothek verzichtet.

2.2.7 Image Registration Toolkit (IRTK)

Hierbei handelt es sich um eine Bildregistrierungs-Software, die im Imperial College London [IRTK] entwickelt wurde. Allerdings wurde die Software 2008 an die Firma IXICO verkauft, die den Quellcode nicht mehr der Öffentlichkeit zur Verfügung stellt. Dementsprechend kam das Toolkit für eine Verwendung in dieser Arbeit nicht in Frage. IRTK ist unabhängig von ITK, die ähnlichen Bezeichnungen sind zufällig.

2.2.8 Medical Imaging Interaction Toolkit (MITK)

Bei **MITK** handelt es sich um ein im deutschen Krebsforschungszentrum Heidelberg (DKFZ) entwickeltes Toolkit, für die Entwicklung von Bildverarbeitungs- und Visualisierungsanwendungen in der Medizin [MITK] .
MITK kombiniert die beiden Bibliotheken ITK und VTK und bietet auf der gleichen Abstraktionsebene

eigens entwickelte Funktionalitäten an. Es nutzt ferner die C++Benutzeroberflächenbibliothek Qt. Es liegen sowohl der Quellcode als auch eine fertig implementierte Applikation vor, in der die Mehrzahl der angebotenen Funktionalitäten zur Verfügung steht. Als Toolkit kann MITK auch in andere Projekte eingebunden und ihre Funktionalitäten genutzt werden. Zudem ermöglicht sie auch das Verarbeiten von nicht-medizinischen Bilddaten und –formaten. Neue Funktionalitäten können als PlugIn implementiert werden, geeignete Rahmenstrukturen werden zur Verfügung gestellt.

2.2.9 MeVisLab

MeVisLab ist eine integrierte Entwicklungsumgebung (IDE) für Applikationen der medizinischen Bildverarbeitung [MeVisLab] . Sie basiert auf einer, unter dem ILAB Projekt (Image Vision Library) vom „CeVis Institut“ entwickelten, plattformunabhängigen C++ Bibliothek namens Mevis Library (kurz ML). Seit 2002 wird MeVisLab als eigenständiges Projekt von Mevis und dem Fraunhofer Institut weiterentwickelt und genutzt. Hier handelt es sich um ein Framework zur Entwicklung von bildbearbeitenden Algorithmen und deren Visualisierung. Es bietet eine eigene Entwicklungsumgebung zur Erstellung von sogenannten Bildverarbeitungsnetzwerken, erlaubt aber auch das Implementieren von neuen Modulen mit Visual C++. MeVisLab beinhaltet einige Komponenten von ITK und VTK, unter anderem können ITK-Registrierungsverfahren wie oben erläutert, nachgebaut werden. MeVisLab basiert ebenfalls auf Qt.
Die Entwicklungsumgebung steht in einer eingeschränkten Version kostenlos zur Verfügung.

3 Analyse der Entwicklungsumgebungen

Die in Frage kommenden Bibliotheken und Toolkits (im Folgenden einheitlich als **Entwicklungsumgebung** bezeichnet) wurden bereits im vorherigen Kapitel erläutert. Auf Basis der Recherchen zu den einzelnen Produkten erwiesen sich die Entwicklungsumgebungen MITK und MeVisLab am geeignetsten, da diese erstrangig die Entwicklung eigener Applikationen unterstützen [Bitter07] .

Im Folgenden wird der Entscheidungsprozess beschrieben, bei dem die für diese Arbeit am geeignetsten erscheinende Entwicklungsumgebung ausgewählt wurde. Der Entscheidungsprozess gliedert sich in drei Phasen, der Anforderungsanalyse an die Software, eine kurze Darstellung des Aufbaus und eine kurze Übersicht über das Einarbeiten mit beiden Entwicklungsumgebungen. Anschließend werden die Ergebnisse der Analyse in einer Tabelle dargestellt und ausgewertet.

3.1 Anforderungsanalyse

Nachfolgend werden die Kriterien erläutert, die bei der Wahl der in dieser Arbeit zu verwendenden Entwicklungsumgebung ausgewählt wurden.

Medizinische Bildbearbeitung und -visualisierung

Grundsätzlich muss die Entwicklungsumgebung die Grundlagen der medizinischen Bildverarbeitung beherrschen. Dazu zählen das Verarbeiten des medizinischen Standardbildformats DICOM, die Integration von Registrierungs-, Filter-, und Segmentierungsalgorithmen sowie verschiedene Möglichkeiten zur Visualisierung der Bilddatensätze und anderen Daten wie Landmarken etc.
Sie sollte wenn möglich auf ITK und VTK basieren, da diese beiden Bibliotheken im wissenschaftlichen Bereich anerkannt sind und bereits eine Vielzahl an Funktionen bieten.

Anforderungen an die Architektur

Bei der gewählten Entwicklungsumgebung handelt es sich um eine Gesamtanwendung bei der ein Großteil der in den Grundlagen erläuterten Funktionen zur Verfügung stehen sollte. Zudem sollte sie das Einbinden von externen Bibliotheken möglich sein. Das Entwickeln sollte auf einer möglichst hohen Abstraktionsebene stattfinden, sodass Einarbeitungs- und Entwicklungszeit sich auf ein Minimum reduzieren. Eine Undo/Redo Funktion ist ebenfalls wünschenswert. Ebenso sollte die Architektur gewisse Standards erfüllen wie etwa ein Datenexport von numerischen Daten (Landmarken, etc.) in XML-Format.

Wiederverwendbarkeit

Einmal implementierte Strukturen und Algorithmen sollten in einem anderen Kontext, auch von einem anderen Entwickler, wiederverwendbar sein. Die während dieser Arbeit implementierte Applikation soll beliebig erweiterbar sein.

Updates und Wartung

Das zu wählende Framework sollte kontinuierlich weiterentwickelt werden, dazu gehören regelmäßige Updates und Anpassungen an aktuelle Programmierstandards. Projekte von Forschungsinstituten oder Universitäten haben den Vorteil dass diese ihre Projekte immer auf dem neusten wissenschaftlichen Stand zu halten versuchen.

Support, Dokumentation und Community

Einen guten und schnellen Support sowie eine übersichtliche und ausführliche Dokumentation sind für ein zügiges und effektives Entwickeln essentiell. Eine ausgeprägte Community kann bei Fehlern und Problemen im Rahmen der Implementierung aushelfen.

Bedienbarkeit

Die Einarbeitungszeit für den Programmierer wie für den späteren Anwender soll so kurz wie möglich sein. Sowohl Entwicklungsumgebung wie die spätere Applikation sollten daher intuitiv bedienbar sein.

Kosten

Grundsätzlich sollte es sich bei dem Framework um Open-Source oder kostenlos zugänglichen Code handeln. So sind das aktuell sowie nachfolgende Projekte nicht von finanziellen Mitteln abhängig.

3.2 Softwarearchitektur von MITK und MeVisLab

Im Folgenden wird auf die Softwarearchitektur und das Entwicklungsprinzip der beiden Entwicklungsumgebungen MITK und MeVisLab eingegangen**.**

MITK ist eine C++ basierte Entwicklungsumgebung und kapselt die Verfahren der ITK- und VTK-Bibliotheken. Zusätzlich wird deren Funktionsumfang um einige zusätzliche Funktionalitäten erweitert. Diese werden im sogenannten Toolkit zusammengefasst und bieten den Entwickler unterstützende Features an. Dazu gehören u. A. eine Zusammenstellung von Viewern, ein Undo/Redo Konzept sowie eine vorgegebene Datenstruktur und entsprechende Import/Export Funktionalitäten.
MITK wird dauerhaft weiterentwickelt, es wird empfohlen den aktuellen Entwicklungsstand über die Entwicklungsrepositories zu beziehen [Wiki1] .

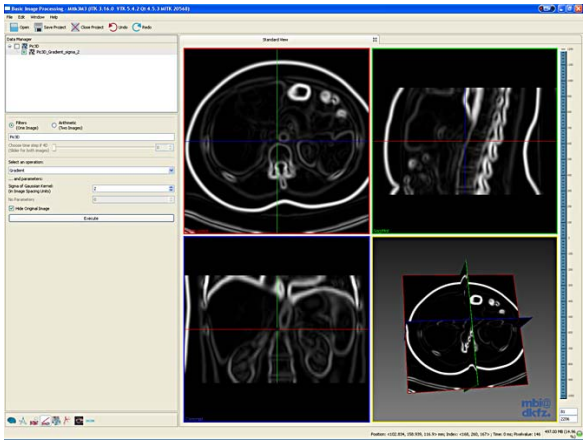


Abbildung 16: Screenshot des mit MITK entwickelten DICOM-Viewers MITK 3M3 [MITK1]

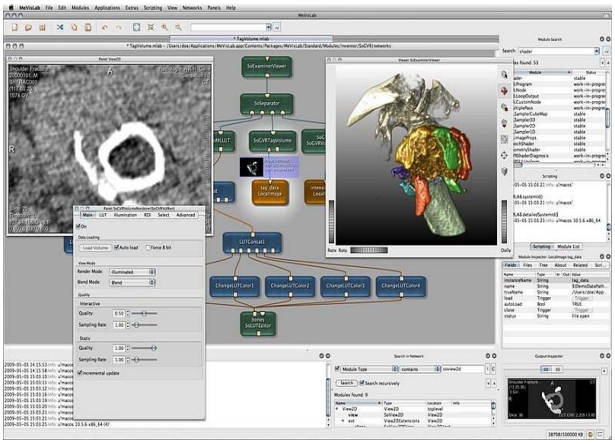


Abbildung 15: Screenshot der MeVisLab Entwickleroberfläche [MeVisLab]

MeVisLab arbeitet mit dem sogenannten **Visual Programming**-Ansatz, bei denen Module (funktionale Einheiten mit Aufgaben) zu bildverarbeitenden Netzwerken gekoppelt werden. Zudem werden in einer eigenen Skriptsprache Benutzeroberflächen generiert und mit Python wird die Neuentwicklung dynamisiert (s. Kapitel 3.5).

Der Vorteil beim Implementieren mit MeVisLab liegt darin, dass das Entwickeln und Testen der Anwendung ohne vorheriges Kompilieren durchgeführt werden kann. Dies schlägt sich auch auf die Gesamtentwicklungszeit nieder (s.

Tabelle 1). Es ist auch möglich konventionell mit C++ zu entwickeln. Die Community trägt viel zu den vorhandenen Modulen bei, die Weiterentwicklung ist sehr konsequent. Dies bestätigen hinzugezogene Quellen die die Anzahl der Module im Jahr 2006 auf 300 beziffern [Bitter07] , 2007 auf 500 Module [Bühler07] und in 2011 ist von 1300 offiziellen Modulen die Rede [MevRef] . Neue Module können sofort von allen in MeVisLab entwickelten Applikationen verwendet werden.

3.3 Einarbeitungsphase der Entwicklungsumgebungen MITK und MeVisLab

Die Einarbeitungszeit in die Entwicklungsumgebung spielt für spätere Anwender und Entwickler eine große Rolle. Daher sind eine ausführliche Dokumentation und detaillierte Tutorials ein Positivum für die zu wählende Entwicklungsumgebung.

3.3.1 MITK

Die Installation von MITK war relativ komplex, besonders die Installationen von verschiedenen benötigten Bibliotheken, sowie die anschließende Verlinkung untereinander und die Parametrisierung der Entwicklungsumgebung VisualStudio, war sehr zeitaufwendig. Es bedurfte einiger Installationsanläufe bis sich die zur Verfügung gestellten Testprojekte fehlerfrei öffnen ließen. Die eben erwähnten Testprojekte waren Bestandteil eines gut erläuterten Tutorials, anhand dessen sich sehr schnell eine Applikation zum Laden und Sichten von Bilddaten implementieren ließ, solange man die Anweisungen strikt befolgte. Am Ende des Tutorials besaß man eine gut funktionierende Applikation für das Laden, Sichten und Segmentieren von Bilddaten, dafür aber recht wenig Einsicht in die Struktur von MITK. Für den anschließenden Versuch, eine eigene Applikation zu entwickeln, fehlten gewisse Dokumentationsschritte. Da MITK am Standort Heidelberg entwickelt wird, gab es allerdings eine sehr große Anzahl an Ansprechpartnern in unmittelbarer Nähe und aus allen Fachgebieten die hierbei weiterhelfen konnten.

Insgesamt erschien das Entwickeln mit MITK sehr komplex und abstrakt, was aber auf die Größe der Bibliothek und den konventionellen Programmieransatz zurückzuführen ist. Das niedrige Abstraktionsniveau erlaubt auf einer Seite sehr viele Freiheiten beim Entwickeln von Algorithmen, erhöht auf der anderen Seite aber den Einarbeitungs- und Entwicklungsaufwand.

3.3.2 MevisLab

Nach einer zügigen Installation verwies ein Popup-Fenster beim Starten von MeVisLab auf verschiedene Tutorials, Beispielnetzwerke und Dokumentationen. Das Eingangstutorial erläuterte u. A. die genaue Datenstruktur von MeVisLab, die Entwicklung von Netzwerken und aller Modultypen. Das Entwickeln von Netzwerken und die Bedienung von MeVisLab verlief sehr intuitiv, direkt nach dem Durcharbeiten des entsprechenden Kapitels im Tutorial ließen sich komplexe bildverarbeitende Netzwerke zusammenstellen und sofort testen.
Für die Entwicklung neuer Module stand ein Wizard zur Verfügung, der die Rahmenstruktur für das

entsprechende Modul bereitstellte. Die Implementierung eines C++ Moduls anhand des Tutorials verlief problemlos, auch ein im Anschluss selbstentwickeltes Modul ließ sich in relativ kurzer Zeit implementieren. Nach dem Kompilieren des Quellcodes stand das Modul gleich zur Verfügung. Weiterführende Schritte im Bereich der Applikationsentwicklung waren allerdings durch das Fehlen einer entsprechenden Lizenz vorerst nicht möglich.

Am Standort Heidelberg arbeitet eine weitere Forschungsgruppe mit MeVisLab zusammen, sodass sich auch für diese Entwicklungsumgebung Ansprechpartner fanden.

Auf Grund des intuitiven Entwicklungsprinzips von MeVisLab war es nach kurzer Zeit möglich, komplexe bildverarbeitende Netzwerke zusammenzustellen. In diesem Zeitraum waren zudem bereits mehrere Modulprototypen entstanden, das Prinzip der Benutzeroberflächengestaltung angerissen worden und es bestand reger Kontakt mit verschiedenen Entwicklern.

Da die während der Einarbeitung entwickelten Applikationen nicht miteinander verglichen werden konnten bezüglich des Faktors Zeitaufwand/Funktionalität, wird hierzu aus [Bitter07] die Tabelle 1 hinzugezogen. Aus der Gesamtzeit geht hervor, dass eine Entwicklung mit MITK auf Basis einer ITK Klasse ca. fünfmal so viel Zeit beansprucht wie eine vergleichbare mit MeVisLab. Dies wird in der Auswertungstabelle (

Tabelle 1) unter dem Faktor **Entwicklungsaufwand** berücksichtigt.

MITK	MeVisLab	Zeit zum Entwickeln der Applikation
N/A	00:00:03	Wrappen einer ITK Klasse
00:10:00	00:00:25	Verbinden des Ein- und Ausgangs der ITK Klasse / des Moduls
05:00:00	00:55:00	Zeit zur Implementierung der Applikation, ohne Wrappen
05:00:00	**01:00:00**	**Gesamtzeit zur Implementierung der Applikation**
00:05:00	00:05:00	Zeit bis die Applikation mit neuen Daten läuft

Tabelle 1: Aufstellung der Entwicklungszeiten spezifizierter Applikationen. Entnommen aus [Bitter07]

3.4 Auswertung

Für die endgültige Auswertung wurden verschiedene Faktoren beachtet, zunächst die Auswahl an bildverarbeitenden Verfahren sowie die technischen Anforderungen und Möglichkeiten. Dazu kamen Faktoren die für die Entwicklung wichtig erschienen, wie solche für den späteren Anwender. Diese werden in der folgenden

Tabelle 2 dargestellt und bewertet.

MeVisLab	MITK	Faktor	Quellen
		Algorithmen	
+	+	Anzahl der bildverarbeitenden Algorithmen	1,2
+	+	Anzahl der registrierenden Algorithmen	1,2
+	+	ITK und VTK werden eingebunden	3
+	-	Interaktives Setzen von Segmentierungsparametern	3
+	-	Interaktives Prototyping von Registrierungsverfahren	3
+	o	Anzahl der vom Anwender konfigurierbaren Algorithmen	3
		Technische Daten	
+	o	Maximale Bildgröße	3
+	+	Betriebssysteme Windows, Linux, Mac	1,2,3
+	+	Unterstützt 64-Bit Systeme	1,2
+	o	Entwicklungsaktivität [1]	1,2
		Implementierung / Entwicklung	
+	o	Entwicklungsaufwand / -zeit	3,4
+	-	Visueller Programmieransatz	1,2
+	+	Undo / Redo Funktion	1,2
+	o	Wiederverwendbarkeit von Codebestandteilen	1,2,3
+	+	Datenexport u.A. in XML	1,2
+	+	GUI für die Bedürfnisse des Anwenders optimierbar	3
-/o	+	Open Source	1,2
+	+	Dokumentation, Wartung, Community	1,2
		Anwendung	
+	+	Expertise benötigt zum Anwenden der entwickelten Software	1,2
18	8	**Ergebnis**	

Tabelle 2: Gegenüberstellung der beiden Entwicklungsumgebungen MITK und MeVisLab.
Die folgenden Quellen werden verwendet: 1) [MeVisLab] 2) [MITK] 3) [Bitter07] 4) Tabelle 1

Das Ergebnis der Auswertungstabelle spricht eindeutig für die Verwendung der Entwicklungsumgebung **MeVisLab** im Rahmen dieser Diplomarbeit.

[1] Bewertet zum Zeitpunkt der Analysephase August 2010. Letzter Release MITK Dez. 2009, letzter Release MeVisLab Juni 2010

3.5 Eingesetzte Tools

Die zu entwickelnde Anwendung wird wie in Kapitel 3.4 auf MeVisLab basieren. Für die Registrierung wird das Registrierungstoolkit MatchPoint über die Bridge integriert.

Die im Rahmen dieser Arbeit verwendeten Tools sind in

Tabelle 3 aufgeführt.

Name	Beschreibung	Liz.
MeVisLab Version 2.1 (64 Bit)	Entwicklungsumgebung	f
Microsoft VisualStudio 2008 (64 Bit)	Entwicklungsumgebung für C++ Module, Algorithmen und die Bridge	k
CMake		f
MatchPoint 0.9	Registrierungsbibliothek auf Basis von ITK	f
Bridge 0.2	Schnittstelle zwischen MatchPoint und MeVisLab	f
Qt 4.6.2	Bibliothek für Benutzeroberflächenbestandteile	o
Python 2.6	Programmiersprache	f

Tabelle 3: Verwendete Bibliotheken und Entwicklungsumgebungen. k = kommerziell, f = frei zur Nutzung, o = Open Source

4 MeVisLab

Die Bildverarbeitungsumgebung MeVisLab arbeitet wie schon beschrieben über den Visual Programming Ansatz (visuelles Programmieren), bei dem per Drag- and-Drop-Verfahren bildverarbeitende Netzwerke zusammengeschlossen werden. Dies hat den Vorteil, dass auch Anwender ohne Programmierkenntnisse mit der Entwicklungsumgebung arbeiten können.

Neben der Entwicklung von Netzwerken unterstützt MeVisLab die Generierung von bildverarbeitenden Applikationen.
Für Entwickler stehen dafür vier verschiedene Lizenzen zur Verfügung, die jeweils unterschiedliche Funktionen von MeVisLab freischalten.
Ohne Lizenz ist es möglich Netzwerke und Module zu entwickeln und bis zu 15 Third-Party oder eigene Module einzubinden. Das Scripten von Methoden beschränkt sich auf kurze, einzeilige Codesegmente. Es ist nicht möglich eigene Module anderen Entwicklern zur Verfügung zu stellen.
Mit der **Evaluationslizenz** ist es möglich, komplexe Methoden an die Module zu koppeln und unbegrenzt Third-Party Module in das Netzwerk einzufügen.
Die **Entwicklerlizenz** (kommerziell oder nicht-kommerziell) erlaubt das „Signieren“ der Module. Das heißt, dass die Module mit dem Lizenzschlüssel versehen werden und so anderen Anwendern ohne Lizenz zur Verfügung gestellt werden können. Eine solche unbefristete nicht-kommerzielle Lizenz kostet für 3000 € pro Arbeitsplatz.
Die **ADK-Lizenz** (Applikation Development Kit) erlaubt es, aus dem entwickelten Netzwerk eine eigenständige Anwendung zu generieren. Diese Lizenz steht nur Mitarbeitern vom Frauenhofer Institut oder wissenschaftlichen Partnern zur Verfügung.

Für den Zeitraum der Diplomarbeit wurde von MeVisLab eine auf sechs Monate befristete Entwicklerlizenz zur Verfügung gestellt.

Im Folgenden werden die Datenstrukturen von MeVisLab erläutert. Anschließend wird anhand eines Beispiels das Arbeiten mit der Entwicklungsumgebung vorgestellt.

4.1 Datenstrukturen

Die Datenstrukturen von MeVisLab werden auf zwei Ebenen betrachtet, einmal die der Fields die alle Parameter umfassen, und die der Module, die Algorithmen, Parameter etc. kapseln. Diese werden im Folgenden erläutert.

4.1.1 Datenfelder (Fields)

Alle Datentypen leiten sich von der virtuellen Basisklasse **Field** (im Folgenden „Feld“ genannt) ab. Diese Klasse erlaubt das Kapseln, Verwalten und Überwachen dieser Parameter. Dabei wird zwischen zwei Feldtypen unterschieden, jene für die komplexen, von außen an das Modul herangetragenen Datentypen, sowie jene die für die primitiven Daten innerhalb des Moduls stehen. Erstere werden in Zukunft unter Konnektorfelder, letztere unter Parameterfelder zusammengefasst.

Konnektorfelder

Die drei Typen der Konnektorfelder umfassen Bildobjekte-, Open Inventor (Kapitel 4.1.2.2) und Base Objekte. Diese Datentypen können nur von außen über vordefinierten Schnittstellen dem Modul übergeben werden, den sogenannten Konnektoren (s. Tabelle 4). Es handelt sich hierbei um komplexe Datenstrukturen, die jeweils nur mit dem gleichen Typ Konnektor verbunden werden können.

Unter den **Bildobjekten** versteht man die interne Repräsentation eines Bildes. Über diese Schnittstelle können alle in MeVisLab geladenen Bilder weitergegeben werden, unabhängig von Dimension, Typ und Größe des Bildes.
Bei **OpenInventor Objekten** (auch als Nodes bezeichnet) handelt es sich um Objekte zur Visualisierung im dreidimensionalen Bereich.
Zu den **Base**-Objekten zählen alle Objekte oder Listen von Objekten die von der MeVisLab Basisklasse „Base“ abgeleitet sind und an andere Module weitergegeben werden sollen. Hierunter fallen die sogenannten Marker oder Listen aus Markern, Farbpalletten, etc.

Konnektortypen	In	Out
Bild Objekte		
OpenInventor Objekte		
Base Objekte		

Tabelle 4: Konnektortypen der verschiedenen Objekte

Parameterfelder

Die Parameterfelder behandeln alle in MeVisLab vorkommenden primitiven Datentypen, wie sie beim Programmieren Verwendung finden, wie Booleans, Strings, Integer, etc. An diese primitiven Felder sind zudem vordefinierte Oberflächenelemente gekoppelt die in Kapitel 4.2.3 näher erläutert werden.

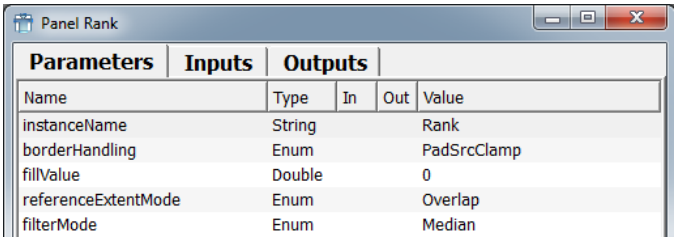


Abbildung 17: Automatic Panel des Moduls Rank

Grundsätzlich werden alle im Modul definierten Parameter nach außen mit dem sogenannten **Automatic Panel** zur Verfügung gestellt (s. Abbildung 17). Dieses Fenster listet alle Parameter samt ihrem Typ und ihrem Wert auf.
Alle Felder werden durch einen sogenannten **FieldListene**r überwacht. Bei jeder Änderung des internen Wertes wird ein Ereignis ausgelöst was etwaige, an das Feld gekoppelte Ereignisse oder andere Felder, ebenfalls auslöst. Die später erklärten Connections (oder Verbindungen) mit denen Parameter im Netzwerk miteinander verbunden werden, sind spezielle Formen dieser FieldListener.

4.1.2 Module

Bei den Modulen handelt es sich um funktionale Einheiten, in denen Algorithmen, Funktionen, Parameter und Daten gekapselt werden. Sie werden durch ihren Typ und die Art ihrer Ein- und Ausgänge nach außen hin definiert. Zu unterscheiden sind die Typen **ML** (kurz für **M**evis **L**ibrary), die **OpenInventor**-Module sowie die **Makro**-Module, in denen andere Module in Netzwerken zu logischen Einheiten verknüpft werden. Diese Module werden im Einzelnen in den folgenden Kapiteln erläutert.

Im Folgenden werden die verschiedenen Modultypen erläutert.

4.1.2.1 ML-Module

Die Buchstaben ML stehen für die **Mevis Library**, die plattform-unabhängige C++ Bibliothek auf der MeVisLab basiert.
Diese Module werden manchmal unter dem Titel „Bildverarbeitungsmodule“ zusammengefasst, was aber irreführend ist, da darunter auch Datencontainer fallen, Module zum Auslesen von DICOM-Headern und zum Laden von Bildern, für mathematische Berechnungen oder verwaltende Funktionen.
Generell werden alle in C++ implementierten Module unter dieser Rubrik zusammengefasst, in MeVisLab wird für sie die Farbe Blau verwendet.

Eine wichtige Untergruppe zu den ML-Modulen bilden die ITK- und VTK-Module. Hierbei handelt es sich um gekapselte original ITK und VTK Klassen. Ein ITK-Registrierungsalgorithmus in MeVisLab besteht aus den gleichen Komponenten, wie sie in Kapitel 2.2.4 erläutert wurden.

4.1.2.2 Open Inventor-Module

Hierbei handelt es sich um Module die auf der Basis von dem 3D Toolkit **Open Inventor** implementiert wurden. Diese Module erlauben das Bearbeiten und Darstellen sowohl der Bilddaten als auch Open Inventor-Objekten. Außerdem ermöglichen sie Interaktionen auf den Bilddaten wie Maus-Events, Tasten-Events etc. Typische Funktionen eines OpenInventor-Moduls wären geometrische Objekte, Kameras, Lichtquellen oder 3D-Viewer.

4.1.2.3 Makromodule

Bei den Makromodulen handelt es sich um Module die andere Module (auch weitere Makromodule) zu einer logischen Einheit zusammenfassen. Dies geschieht durch das Einrichten eines internen Netzwerks. Mit Hilfe von Makromodulen können komplizierte Netzwerke gekapselt oder bestimmte Abfolgen von Funktionen wiederverwertet werden. Sie werden in MeVisLab in der Farbe braun dargestellt. Mit Makromodulen können nur bereits vorhandene Funktionalitäten genutzt und kombiniert werden. Die Benutzeroberfläche wird, analog zu denen von Netzwerken, mit Hilfe der **Module Definition Language** (Kapitel 4.2.3) implementiert.

Es gibt drei Gründe Netzwerke oder Netzwerkbestandteile zu Makromodulen zu kapseln. Einerseits werden durch das Kombinieren von verschiedenen Modulen neue Funktionalitäten entwickelt. Oft verwendete Netzwerkbestandteile werden ebenfalls gekapselt, da sie so einfacher neuen Netzwerken hinzugefügt werden können. Zudem ist es über das Makromodul möglich, kompaktere Benutzeroberflächen zu gestalten.

4.1.3 Bilddatenformat

Um eigene dreidimensionale Datensätze innerhalb von MeVisLab nutzen zu können, müssen sie in das interne DICOM/TIFF Format umgewandelt werden ([MeVisTut10]). Dabei bleiben die Metainformationen der Bilddaten im DICOM-Header erhalten, die Bilddaten werden in das TIFF Format konvertiert. Ohne diese Konvertierung kann MeVisLab die Datensätze nur schichtweise in die Anwendung laden.

4.2 Entwicklungsworkflow

Die Implementierung mit MeVisLab basiert auf der Erstellung sogenannter Netzwerke. Dies geschieht durch das Verbinden von Modulen (Komponenten mit Aufgaben) zwischen vordefinierten Ein-und Ausgängen. Mit MeVisLab lässt es sich, je nach Programmierkenntnissen oder Aufgabenkomplexität, auf drei unterschiedlichen Ebenen arbeiten:

Auf der **visuellen Ebene** lassen sich per Plug&Play-Prinzip bildverarbeitende, Visualisierungs- und Verwaltungsmodule zu einem komplexen Netzwerk zusammenschließen. Der Datenfluss lässt sich an jedem Modulein- und ausgang überprüfen.
Auf der **Scripting-Ebene** können Funktionalitäten und logische Einheiten zu Netzwerken oder Makromodule (Kapitel 4.1.2.3) zusammengefasst werden. Dies erlaubt das Kapseln von komplexen Funktionalitäten und erlaubt eine Wiederverwendung dieser Subnetzwerke in anderen Projekten. Auf dieser Ebene werden außerdem die Benutzeroberflächen definiert.
Auf der **C++ Ebene** werden neue Algorithmen und Datenstrukturen implementiert. MeVisLab bietet für diese Module eine automatisch generierte Codevorlage sowie mehrere Bibliotheken.

Der normale Arbeitsablauf zur Neuentwicklung [MeVisTut10] , Kapitel 2.2 einer Applikation wäre der folgende:

1. Vorhandene Module zu einem Netzwerk zusammenfassen Kapitel 4.2.1
2. Entwicklung von neuen Modulen Kapitel 4.2.2
3. Definition der Benutzeroberfläche Kapitel 4.2.3
4. Kapseln von wiederverwendbaren Funktionalitäten zu Makromodulen Kapitel 4.2.4
5. Scripting von Methoden und Benutzeroberflächen für Applikationen, Module Kapitel 4.2.5

Um diesen Ablauf exemplarisch darzustellen, wird im Folgenden die Entwicklung einer einfachen Applikation erläutert.

4.2.1 Vorhandene Module zu einem Netzwerk zusammenfassen

Das Minimum was eine Bildverarbeitungsapplikation zur Verfügung stellen sollte ist ein Modul zum Laden der Bilddaten, eines zum Bearbeiten und eines zum Visualisieren. Umgangssprachlich spricht man bei dieser Konstellation von einer Pipeline [MeVisTut10] .

Basis jeder Bildverarbeitung ist das Bild, daher wird zuerst das Modul **LocalImage** hinzugefügt. Durch Doppelklick auf das Modul öffnet sich dessen Benutzeroberfläche. Über diese lässt sich ein beliebiger Testdatensatz auswählen (s. Abbildung 18, links).

Um den Bilddatensatz zu betrachten muss dem Netzwerk noch ein Viewer hinzugefügt werden, in diesem Fall ein einfacher zweidimensionaler Viewer namens View2D, dargestellt in Abbildung 18 rechts.

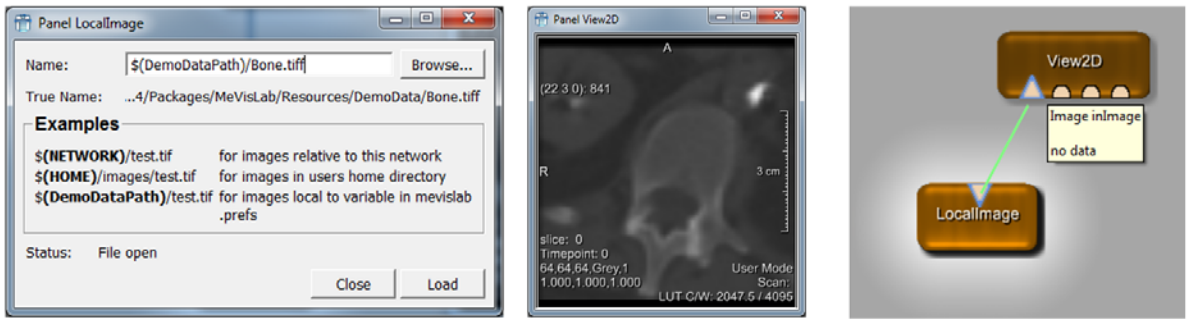


Abbildung 18: links – Screenshot des LocalImage Moduls zum Auswählen der darzustellenden Bilddatensätze
mittig – Screenshot des Viewers
rechts – Erste Netzwerkverbindung

Mit der Maus wird der Bildausgang des LocalImage-Moduls markiert und eine dauerhafte Verbindung zum Bildeingang des Viewers gezogen. Durch Doppelklick auf das Viewer-Modul öffnet sich dessen Benutzeroberfläche und das Bild ist sichtbar, wie in Abbildung 18 mittig zu sehen. Dieser Viewer bietet bereits verschiedene Funktionalitäten an, wie das Scrollen durch die verschiedenen Bildschichten oder das Zoomen.

Zu einem realistischen Bildverarbeitungsnetzwerk gehört weiterhin noch ein verarbeitendes Modul das zwischen den Viewer und des Lade-Modul geschaltet werden muss. Also wird die Verbindung zwischen den beiden Modulen vorerst wieder getrennt.

Es wird nun ein Gauß-Filter für die Bildglättung zum Netzwerk hinzugefügt und entsprechend mit den anderen Modulen verbunden. Im Fenster vom Viewer erscheint nun ein bereits gefiltertes Bild. Ebenfalls per Doppelklick auf das Modul wird die Benutzeroberfläche des Filters geöffnet. Die verschiedenen Filterparameter können über diese Oberfläche verändert werden und das Ergebnis ist sofort auf dem darstellenden Viewer zu sehen (s. Abbildung 19).

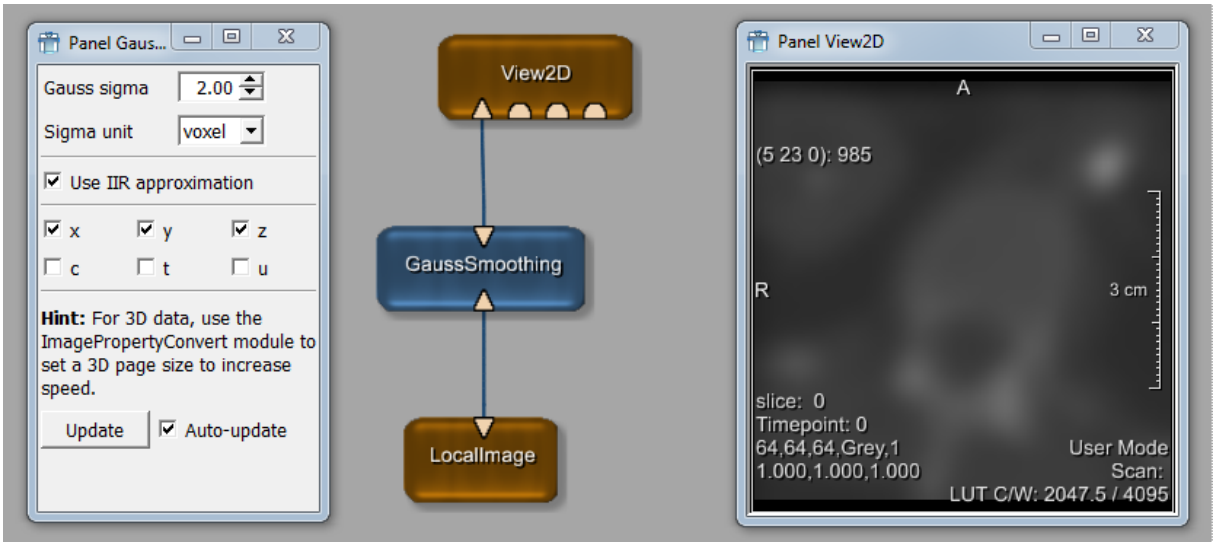


Abbildung 19: Einfache Bildverarbeitungs-Pipeline

Für die Beurteilung eines Filters ist eine Vorher-Nachher-Ansicht des Bildes von Nutzen. Daher wird das ungefilterte Bild vom Bildausgang des LocalImage mit einem neuen Viewer-Modul verbunden. Grundsätzlich gilt, dass Modulausgänge an mehrere andere Moduleingänge gekoppelt werden können, Moduleingänge hingegen können nur mit einem einzigen Modul verbunden werden.

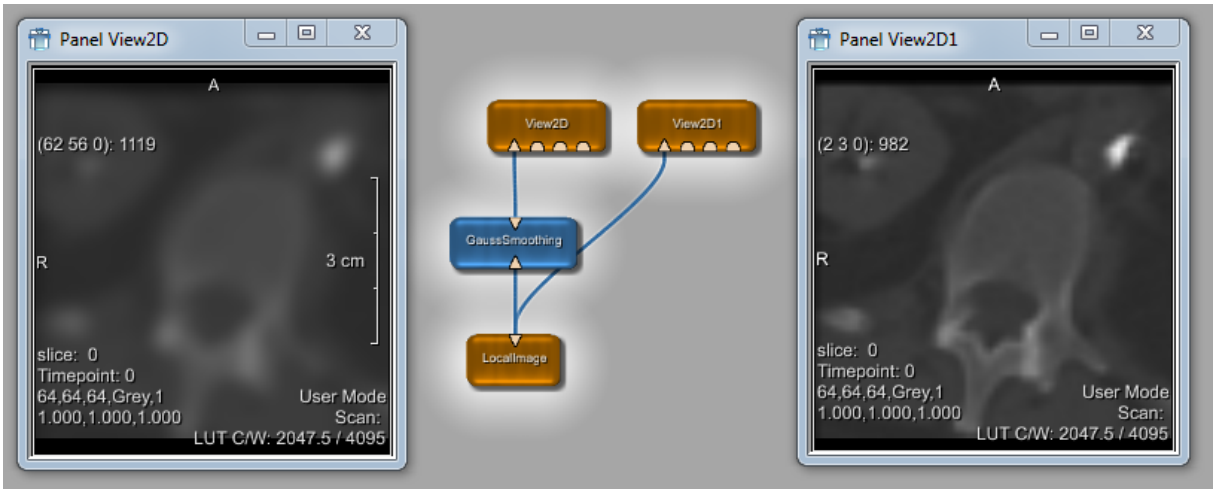


Abbildung 20: Netzwerk mit Vorher-Nachher-Ansicht

Für ein paralleles Sichten der Bilder wäre es außerdem von Nutzen, wenn beide Viewer synchron die gleichen Bildschichten zeigen.

Dafür wird bei beiden Viewern das **Automatic Panel** geöffnet, per Rechtsklick auf das Modul wird ein entsprechendes Menu geöffnet. Über das Automatic Panel werden nun die Parameterfelder **startSlice** der beiden Viewer synchronisiert (s. Abbildung 21). Dazu zieht man jeweils eine Feldverbindung der **Out** Spalte des einen Viewers zur **In** Spalte des anderen. Scrollt man nun durch einen der beiden Viewer, wird der andere zur entsprechenden Schicht synchronisiert.

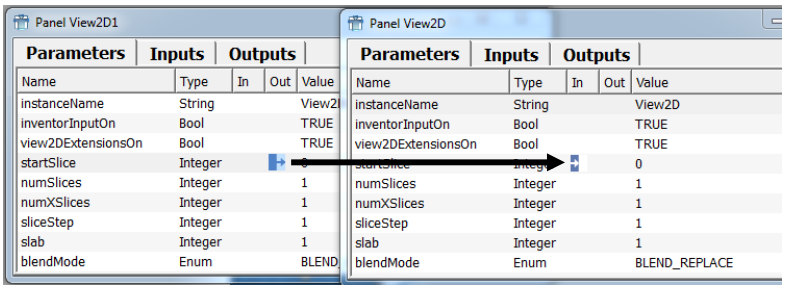


Abbildung 21: Verbinden von Parameterfeldern mit Hilfe des AutomaticPanels

4.2.2 Entwicklung neuer Module

Neue Algorithmen und Prozeduren werden in Form von Modulen entwickelt. Dafür bietet MeVisLab einen Wizard an, der für jeden implementierbaren Modultyp ein Codegerüst generiert. In diesem Beispiel wird ein, die Grauwerte des Bildes invertierendes, Modul benötigt. Da dieses Modul die Bilddaten direkt manipulieren wird, soll es als ML-Modul implementiert werden.

Das Verfahren zum Erstellen eines neuen Projektes mit Hilfe des **Projektwizards** ist in Abbildung 22 dargestellt.

Nach dem Starten des Wizards wird zuerst der Modultyp **ML** ausgewählt (s. Abbildung 22, links oben). Im nächsten Fenster müssen ein eindeutiger Name und verschiedene Metadaten, wie z.B. der Name des Autors, Stichwörter zur Funktionalität, etc. eingetragen werden (s. Abbildung 22, links unten).
Im darauffolgenden Menu werden u. A. die Anzahl der Ein- und Ausgänge definiert und benannt (s. Abbildung 22, rechts unten). Weitere Konfigurationen sind optional, daher kann nun das Codegerüst generiert werden. Die generierten Dateien sind in Abbildung 22 rechts oben dargestellt.

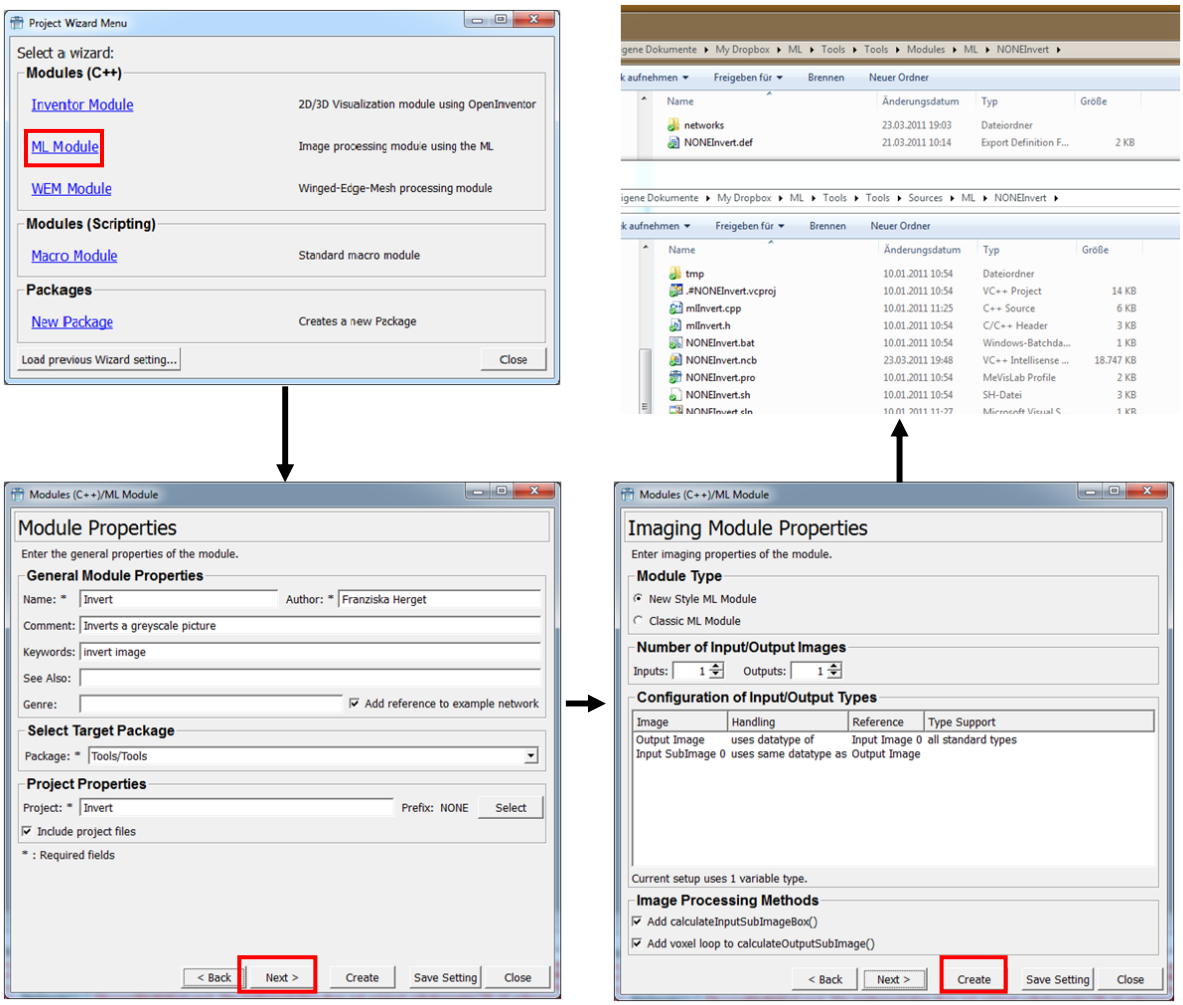


Abbildung 22: Erstellung eines ML-Moduls mit dem Project Wizard.

MeVisLab erstellt nun mehrere Dateien, die es erlauben unter verschiedenen Betriebssystemen und Compilern das Modul zu implementieren. Hierauf wird nun das entsprechende Projekt gestartet.

Bei der **Invertierung** handelt es sich um ein pixelbasiertes Verfahren (s. Kapitel 2.1.3). Das heißt, dass anhand von Schleifen, die über alle Dimensionen des Bildes iterieren, nacheinander auf alle Pixel des Bildes zugegriffen wird. Deren Grauwerte werden mittels einer Funktion in neue Grauwerte überführt und daraufhin in das Ergebnisbild geschrieben.
Die dazugehörige Formel wäre:

$$\text{Pixel}_{\text{Ergebnisbild}}(x_a, y_a, z_a) = \text{MaximalerGrauwert}_{\text{Ausgangsbild}} - \text{Pixel}_{\text{Ausgangsbild}}(x_a, y_a, z_a)$$

Formel 1: Berechnung des invertierten Bildes

Um das Ausgangsbild korrekt in das invertierte Ergebnisbild zu überführen, müssen in dem Projekt die Methoden **calculateOutputImageProperties** und **calculateOutputSubImage** ausimplementiert werden.

In der **calculateOutputImageProperties-**Methode werden Bildeigenschaften wie etwa der minimale und der maximale Grauwert des Bildes neu gesetzt. Im Fall der Invertierung werden diese beiden Werte mit dem jeweiligen Gegenpart überschrieben.

```
1  void Invert::calculateOutputImageProperties (int outIndex, PagedImage* outImg)
2  {
3    ML_TRACE_IN("Invert::calculateOutputImageProperties ()");
4
5    //
6    const double maxValue = getInputImage(0)->getMaxVoxelValue();
7    const double minValue = getInputImage(0)->getMinVoxelValue();
8
9    outImg->setMinVoxelValue(minValue);
10   outImg->setMaxVoxelValue(maxValue);
11
12   // Change properties of output image outImg here whose
13   // defaults are inherited from the input image 0 (if there is one).
14 }
```

Quellcode 1: Setzen der neuen Bildparameter „Maximaler Grauwert“ und „Minimaler Grauwert“

Innerhalb der **calculateOutputSubImage**-Methode wird nun das Ergebnisbild anhand der oben genannten Formel berechnet.

Zunächst werden die entsprechenden Schleifen über die verschiedenen Bilddimensionen (x, y, z, t) implementiert , über die auf jeden der Bildpixel zugegriffen werden kann. Dies erfolgt mit Hilfe des mehrdimensionalen ImageVectors. Dies ist im blau umrandeten Bereich in Quellcode 2 zu sehen. Zur Berechnung des invertierten Bildes wird der maximale Grauwert des Ausgangsbildes benötigt, dieser Wert wird über die Zeile 12 in Quellcode 2 dem Anfang der Methode hinzugefügt. Innerhalb der untersten Schleifenebene soll nun auf die einzelnen Pixel zugegriffen, der neue Grauwert berechnet und in das Ausgangsbild geschrieben werden. Dies erfolgt über die Zeilen 20 – 22 in Quellcode 2, innerhalb des grünen Rahmens.

```
1  void Invert::calculateOutputSubImage (TSubImage<T> *outSubImg, int outIndex
2                                        , TSubImage<T> *inSubImg0
3                                        )
4  {
5    ML_TRACE_IN("template <typename T> Invert::calculateOutputSubImage ()");
6
7
8    const SubImageBox validOutBox = outSubImg->getValidRegion();
9
10
11   ImageVector p;
12   const double maxValue = getInputImage(0)->getMaxVoxelValue();
13
14
15
16     for (p.t=validOutBox.v1.t;  p.t<=validOutBox.v2.t;  ++p.t) {
17         for (p.z=validOutBox.v1.z;  p.z<=validOutBox.v2.z;  ++p.z) {
18           for (p.y=validOutBox.v1.y;  p.y<=validOutBox.v2.y;  ++p.y) {
19             for (p.x = validOutBox.v1.x; p.x <= validOutBox.v2.x;  ++p.x, ++outVoxel, ++inVoxel0){
20                 const T* inVoxel0 = inSubImg0->getImagePointer(p);
21                 T*  outVoxel = outSubImg->getImagePointer(p);
22                 *outVoxel = maxValue - *inVoxel0;
23         }
24       }
25     }
26   }
27 }
```

Quellcode 2: Implementierung des Invertierungsverfahrens.
Blauer Rahmen: Schleife die durch alle Bilddimensionen iteriert
Grüner Rahmen: Invertierungsalgorithmus

Zum Schluss wird das Modul erstellt und steht nach einem Neustart von MeVisLab zur Verfügung.

Ist das Modul fehlerhaft wird es rot dargestellt und seine Verbindungen zu anderen Modulen werden unterbunden um die Restfunktionalität des Netzwerks aufrecht zu erhalten. In diesem Fall muss der Quellcode auf Fehler überprüft werden. Ist das Modul fehlerfrei, kann es wie jedes andere dem Netzwerk hinzugefügt werden.

Um beide Filterverfahren zur Verfügung stellen zu können, wird zwischen den Filtermodulen und dem Viewer ein Switch-Modul (engl. für Schalter) eingefügt. Dadurch wird es dem Entwickler ermöglicht, zwischen den Filtern hin und her zu schalten, ohne das Netzwerk manipulieren zu müssen. Das neue Netzwerk ist in Abbildung 23 dargestellt.

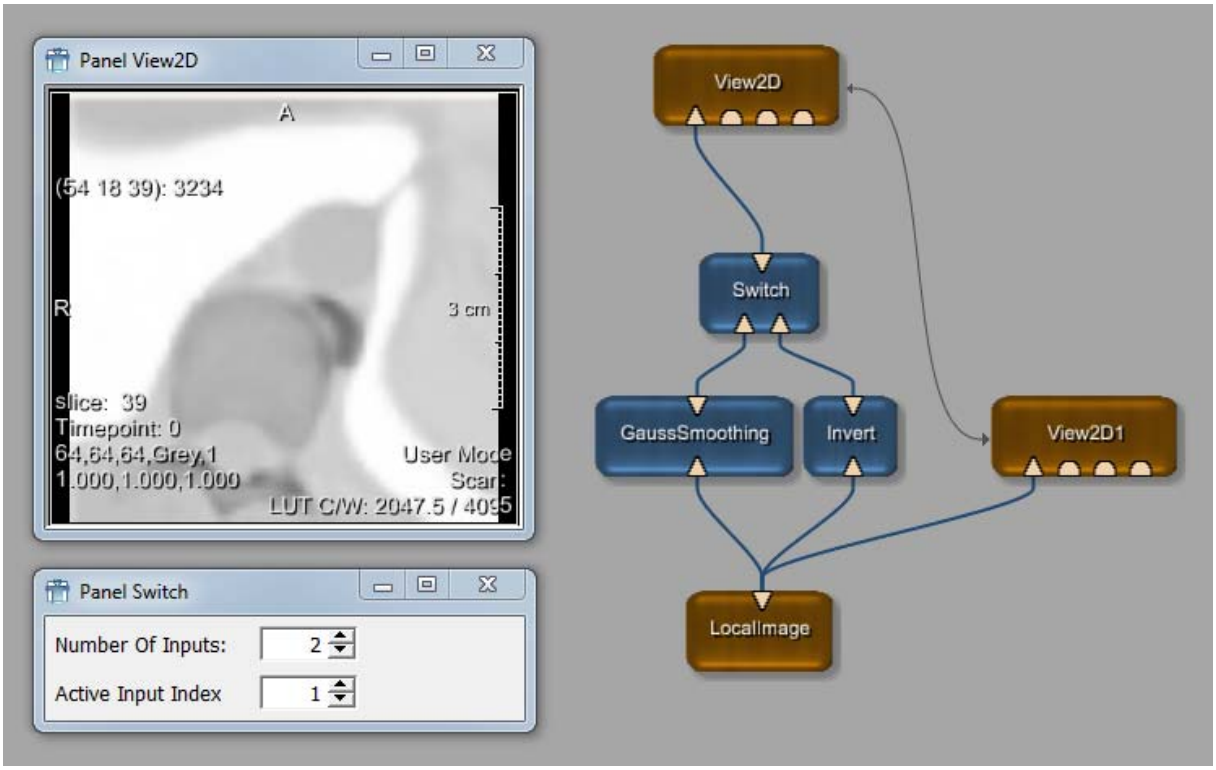


Abbildung 23: Netzwerk erweitert um das Invert und ein Switch-Modul

4.2.3 Definition der Benutzeroberfläche

Jedes Modul besitzt eine eigene Benutzeroberfläche, das heißt das Arbeiten mit diesem Netzwerk wird mit jedem weiter hinzugefügten Modul unübersichtlicher. Um die Übersicht über die verschiedenen Funktionalitäten beizubehalten, wird nun eine passende Benutzeroberfläche für dieses Netzwerk entwickelt.
Mit der **Module Definition Language** (kurz MDL) werden in Mevislab die Benutzeroberflächen geschrieben. Dabei handelt es sich um eine eigene Definitionssprache. Da keine neuen Funktionen innerhalb dieses Netzwerks entwickelt wurden, kann auf die Benutzeroberflächen der verwendeten Module zurückgegriffen und diese in der neuen Oberfläche wiederverwendet werden.

Zuallererst sollen alle Moduloberflächen in einem Fenster dargestellt werden. Dazu wird der MeVisLab-interne Editor geöffnet. Ein neues Fenster wird mit dem Tag **Window** definiert. Diesem wird ein horizontales Layout hinzugefügt, sowie die Benutzeroberflächen aller im Netzwerk verwendeten Module. Dazu wird das Element **Panel** benötigt. Innerhalb dieses Elements werden alle Oberflächenbestandteile des referenzierten Moduls geladen. In Quellcode 3 wird die bisherige Version des Benutzeroberflächen-Skripts dargestellt.

```
Window{
  Horizontal{
    Panel viewerOriginal{module = View2D1}
    Panel viewerGefiltert{module = View2D}
    Panel switch{module = Switch}
    Panel invert{module = Invert}
    Panel gaussSmoothing{module = GaussSmoothing}
    Panel localImage{module=LocalImage}
  }
}
```

Quellcode 3: Erste Quellcodeversion der Benutzeroberfläche

Anschließend wird die neue Applikation gestartet, die momentane Darstellung der Applikation ist in Abbildung 24 zu sehen.

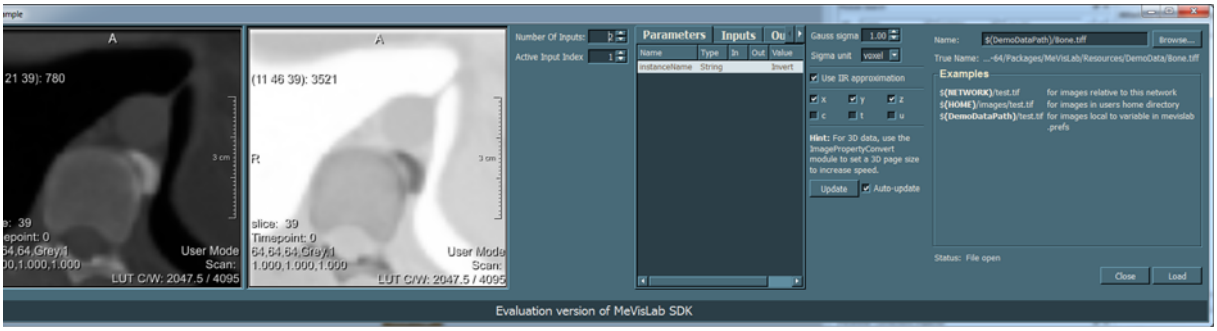


Abbildung 24: Erste Version der Benutzeroberfläche

Wie in Abbildung 24 zu erkennen ist die Benutzeroberfläche nicht sehr übersichtlich, daher werden im Folgenden weitere Modifizierungen an der Benutzeroberfläche vorgenommen.

Vom Modul LocalImage werden nur die Browse-Funktion und der Load-Knopf benötigt, beim Switch reicht es aus sich auf den **ActiveInputIndex** (Bildeingang der weitergereicht wird) zu beschränken. Der Inverter kann komplett entfernt werden, da bei ihm noch keine Benutzeroberfläche definiert wurde und nur das AutomaticPanel dargestellt wird (s. Kapitel 4.1.1.2.).

In Quellcode 4 in Zeile 4 ist zu sehen, wie auf das Parameterfeld **Load** des Moduls LocalImage zugegriffen wird und dieses nun in der eigenen Applikation unter dem Alias **loadImage** zur Verfügung steht.

Anschließend wird der Tag Parameters um den Auswahlindex des Switch und des Load-Buttons des LocalImage Moduls erweitert. Nun müssen diese Komponenten an Benutzeroberflächenelemente gekoppelt und diese in der Benutzeroberfläche der Applikation dargestellt werden. Es bietet sich an die Parameter an die gleichen Oberflächenelemente zu koppeln wie sie schon in den Modulen verwendet wurden. Dazu werden die Felder innerhalb des **Window**-Tags mit den entsprechenden Benutzeroberflächenelementen hinzugefügt.

Anschließend wird mit Hilfe verschiedener Layouts die Benutzeroberfläche neu strukturiert (s. Abbildung 25).

```
Interface {
 Parameters{
   Field browseImage {internalName = LocalImage.name}
   Field loadImage {internalName = LocalImage.load}
   Field switch {internalName = Switch.currentInput}
 }
}

Window{
  Vertical{
   Box Load {
     Field browseImage { browseButton = True }
     Button loadImage {}
   }

   Box Viewer{
    layout = Horizontal
    Panel viewerOriginal{module = View2D1}
    Panel viewerGefiltert{module = View2D}
   }
   Box Filter{
      NumberEdit switch {step = 1}
      Panel gaussSmoothing{module = GaussSmoothing}
    }

  }
}
```

Quellcode 4: Script zur Benutzeroberfläche

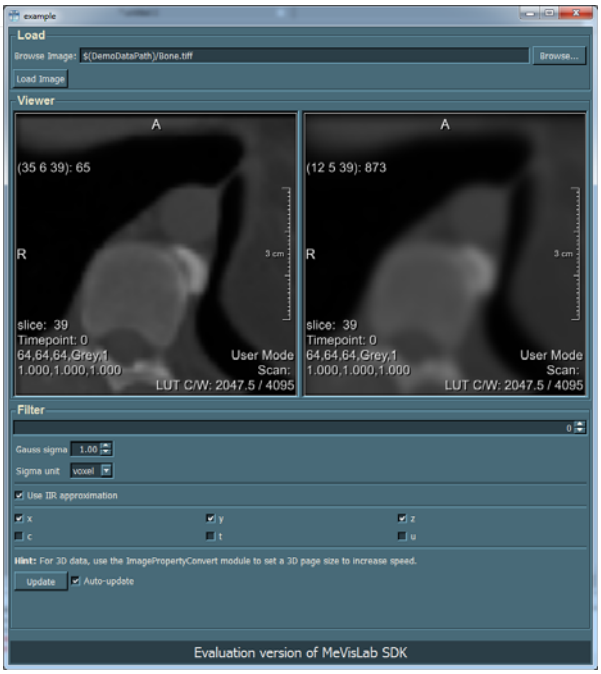


Abbildung 25: Benutzeroberfläche

4.2.4 Kapseln von Funktionen

Wie im Kapitel Makromodule beschrieben bietet es sich an häufig verwendete oder komplexe Netzwerkstrukturen in Makromodulen zusammenzufassen. Im Folgenden werden nun die beiden Filter samt der Vorher-Nachher-Ansicht in ein Makromodul namens **Pipeline** gekapselt.

Dazu wird der Project Wizard erneut aufgerufen und ein entsprechendes Makromodul erstellt. Die ersten beiden Schritte bei der Erstellung eines Moduls sind analog zu denen in Kapitel 4.2.2 durchzuführen. Die generierten Dateien umfassen eine Netzwerkdatei (*.mlab), eine Scriptdatei (*.script) und bei Bedarf eine Python oder JavaScript Datei.

Um das **interne Netzwerk** einzurichten, muss nun die **Pipeline.mlab** Datei in MeVisLab geöffnet werden. Es werden alle Module, bis auf das LocalImage-Modul, aus dem vorher entwickelten Netzwerk kopiert und in das interne Netzwerk des Makromoduls eingefügt. Da auch ein Großteil der vorherigen Benutzeroberfläche übernommen werden kann, wird dessen Code, bis auf die besagten LocalImage-Bestandteile, ebenfalls in die Pipeline.script Datei kopiert (Quellcode 1).

Im nächsten Schritt werden die Modulein- und ausgänge definiert, durch die von außen auf das Modul zugegriffen werden kann.

Ein- und –Ausgänge können bei Makromodulen nur über die Ein- oder Ausgänge interner Module definiert werden. Diese werden nach außen zur Verfügung gestellt. Da nur ein Bildeingang benötigt, aber drei interne Moduleingänge angeschlossen werden müssen, wird dem Netzwerk ein **Bypass**-Modul hinzugefügt und mit den entsprechenden Modulen verbunden (s. Abbildung 26). Dieses Modul reicht die eingehenden Daten weiter und wird dazu verwendet, von einem Punkt aus mehrere Module zu bedienen.
Anschließend wird der Moduleingang des Bypass-Moduls als der Eingang des gesamten Makromoduls definiert. Dazu wird innerhalb des Interface Tags, in der Gruppe Inputs, das entsprechende Feld erstellt. (s. Quellcode 5, Zeile 21).

Da das bearbeitete Bild auch außerhalb des Moduls zur Verfügung stehen soll, wird anschließend noch ein passender Modulausgang definiert (s. Quellcode 5, Zeile 17). Nach einigen Modifikationen an der Benutzeroberfläche erhält man nun nach dem Starten des Moduls **PipeLine** die gleiche Ansicht auf die Daten wie in Abbildung 25 im vorherigen Kapitel.

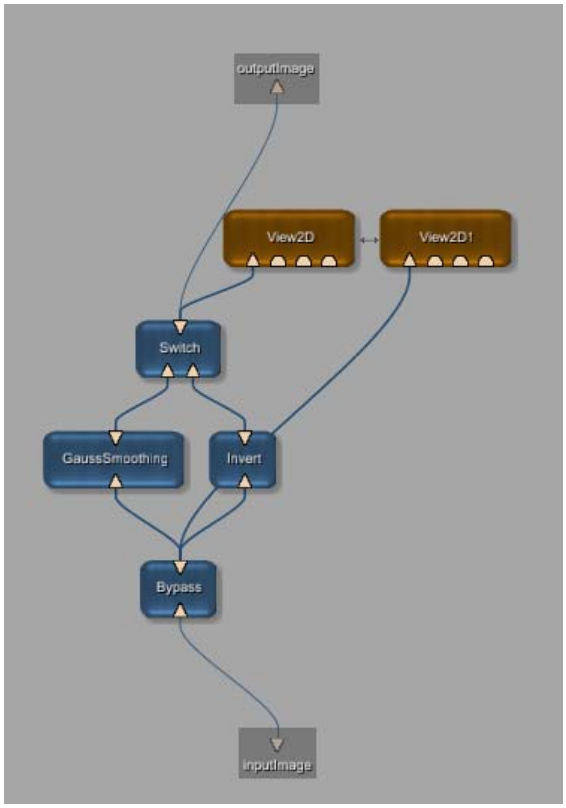


Abbildung 26:Fertiges inneres Netzwerk des Makromoduls Pipeline

```
12 Interface {
13   Inputs{
14     Field inputImage {internalName = Bypass.input0}
15   }
16   Outputs{
17     Field outputImage {internalName = Switch.output0}
18   }
19   Parameters{
20     Field switch {internalName = Switch.currentInput}
21   }
22 }
23
24 Commands {
25   source = $(LOCAL)/Pipeline.py
26 }
27
28 Window{
29   Vertical{
30     Box Viewer{
31       layout = Horizontal
32       Panel viewerOriginal{module = View2D1}
33       Panel viewerGefiltert{module = View2D}
34     }
35     Box Filter{
36         Label{title = "Filtertype"}
37         NumberEdit switch {step = 1 maxw = 50}
38
39       Box "GaussSmoothing"{
40         Panel gaussSmoothing{module = GaussSmoothing}
41       }
42     }
43   }
44 }
```

Quellcode 5: Script für die Benutzeroberfläche des Makromoduls Pipeline

4.2.5 Methodenentwicklung

Der Switch erleichtert zwar den Wechsel zwischen den beiden filternden Modulen, allerdings ist die Anwendung des Switches noch recht umständlich. Zudem ist die Benutzeroberfläche weiterhin unübersichtlich, da nur die Benutzeroberfläche des **GaussSmoothing** Moduls zur Verfügung steht und diese, unabhängig davon ob es über das Switch angesprochen wird oder nicht, modifiziert werden kann.
Diese beiden Probleme werden nun im Folgenden durch das Koppeln an Python gelöst.

Zuallererst wird die Benutzeroberfläche erneut umstrukturiert. Dazu wird der Zähler für den **Switch** entfernt. Für jeden angelegten Filter wird nun ein eigener Container innerhalb eines **TabView-**Elements angelegt. Dies hat den Vorteil dass nur die Benutzeroberfläche des momentan ausgewählten Moduls angezeigt wird (s. Abbildung 27).

Abbildung 27: Unterteilung der Benutzeroberfläche in Tabs für die verschiedenen Filter

Um nun den Switch an den entsprechenden Tab zu koppeln wird nun an jedes **TabView** Element ein **tabSelectedCommand** angefügt. Dies ist in Quellcode 6 in den Zeilen 38 und 42 geschehen. Es werden Methodennamen aus der in der Command Sektion (Quellcode 5, Zeile 24 – 26) definierten Python-Datei angegeben.

```
28 Window{
29   Vertical{
30     Box Viewer{
31       layout = Horizontal
32       Panel viewerOriginal{module = View2D1}
33       Panel viewerGefiltert{module = View2D}
34     }
35     Box Filter{
36       TabView{
37         Box "GaussSmoothing"{
38           tabSelectedCommand = selectGauss
39           Panel gaussSmoothing{module = GaussSmoothing}
40         }
41         Box "Inverter"{
42           tabSelectedCommand = selectInvert
43           Label{title= "This module inverts the given picture"}
44         }
45       }
46     }
47   }
48 }
```

Quellcode 6: Script für die Benutzeroberfläche. In den Zeilen 38 und 42 wurden tabSelectedCommands eingeführt.

Anschließend wird die zugehörige Python-Datei geöffnet und die beiden neu definierten Methoden hinzugefügt und ausimplementiert, zu sehen in Quellcode 7. Die Methoden steuern die Eingänge des Switchs.

```
14 def selectGauss():
15   ctx.field("Switch.currentInput").setValue(0)
16   return
17
18 def selectInvert():
19   ctx.field("Switch.currentInput").setValue(1)
20   return
```

Quellcode 7: Implementierung der Methoden in Python

Damit ist die Umwandlung des bisherigen Netzwerks in das Modul Pipeline fertiggestellt. Die Benutzeroberfläche des fertigen Moduls ist in der Abbildung 28 dargestellt.

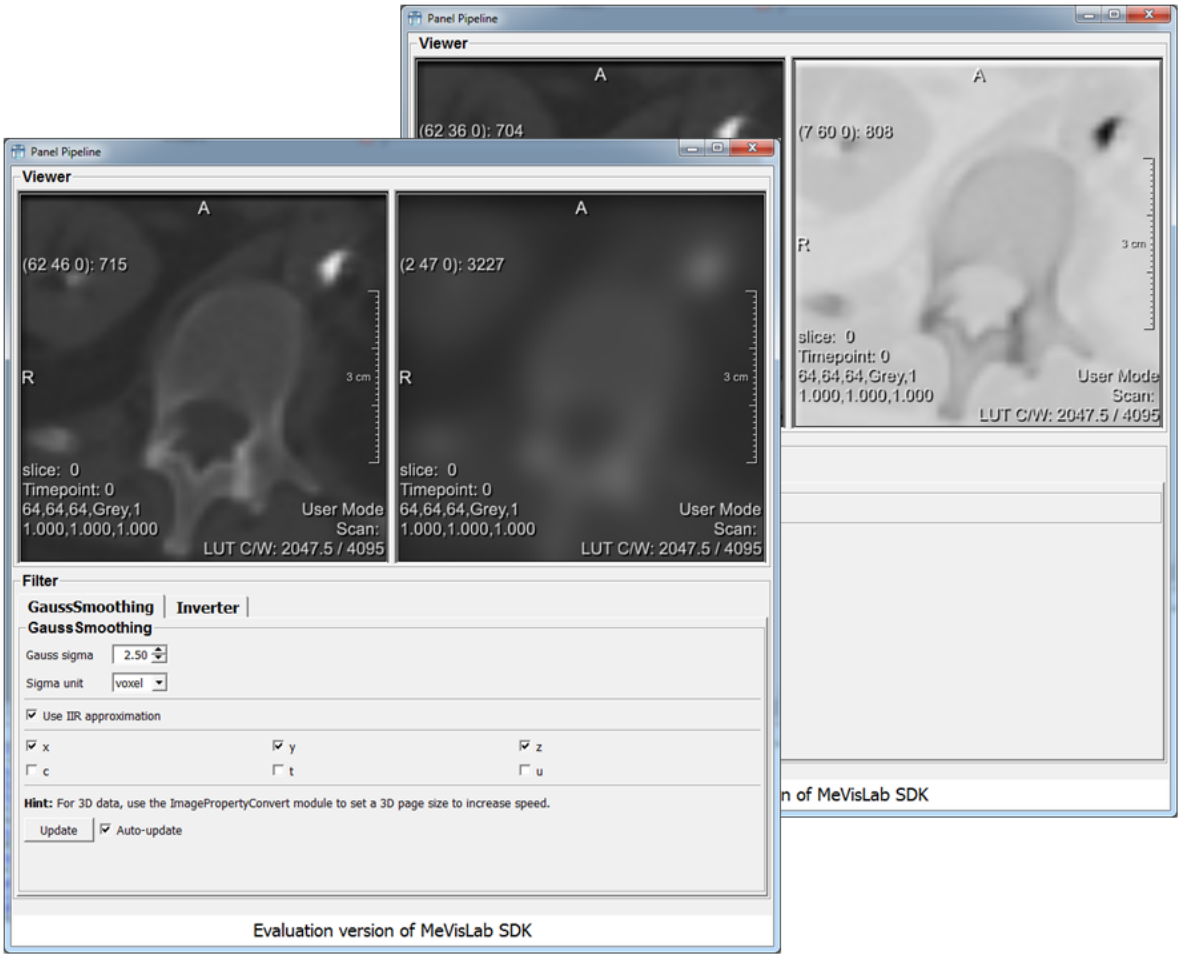


Abbildung 28: Benutzeroberfläche des Moduls Pipeline. Vorne: Aktiver Gauß Filter Hinten: Aktive Invertierung

4.2.6 Makromodul oder Netzwerk veröffentlichen

Gegebenenfalls müssen Module oder Netzwerke noch bearbeitet werden, wenn sie an dritte Personen weitergegeben werden sollen. Dazu gehören das sogenannte Signieren und die Dokumentation.
Falls das Modul an Außenstehende ohne Lizenz weitergegeben werden soll, muss es signiert werden. Dazu wird das Modul **SignFiles** in einem beliebigen Netzwerk gestartet. Unter Angabe der Entwickler-Lizenz werden nun alle entwickelten Komponenten mit einem Code versehen, der das Nutzen und Ausführen der Module und Netzwerkeauch ohne eine entsprechende Lizenz erlaubt. Darüber hinaus ist es möglich für den Nutzer verschiedene Dokumentationen zu hinterlegen. Dazu gehört ein **Beispielnetzwerk**, in dem die praktische Anwendung des Moduls demonstriert wird. Daneben kann man eine Hilfe-Datei erstellen, in die genaue Erläuterungen zur Verwendung, Parametrisierung und Aufbau des Moduls geschrieben werden können. Näheres wird in Kapitel 6.3 erläutert.

5 Anforderungen und Konzeption

Die Konzeptionsphase der zu entwickelnden Applikation umfasst drei Stufen. Aus den Ergebnissen der ersten Stufe, der Anforderungsanalyse, ergeben sich die Aufgaben und die Struktur des Netzwerks (zweite Stufe) und daraus wiederum die Struktur der Benutzeroberfläche (dritte Stufe).

5.1 Anforderungen

Die in Kapitel 2 beschriebenen vor- und nachbereitenden Prozesse der Registrierung können zu der sogenannten Registrierungsprozesskette zusammengeführt werden, dargestellt in Abbildung 29. Dabei wurden alle für das Prä- und Postprozessing der Registrierung benötigten Funktionalitäten in sieben Kettenglieder aufgeteilt. Neben diesen Prozessen bestehen noch Anforderungen an die Erweiterbarkeit und an das Loggings.

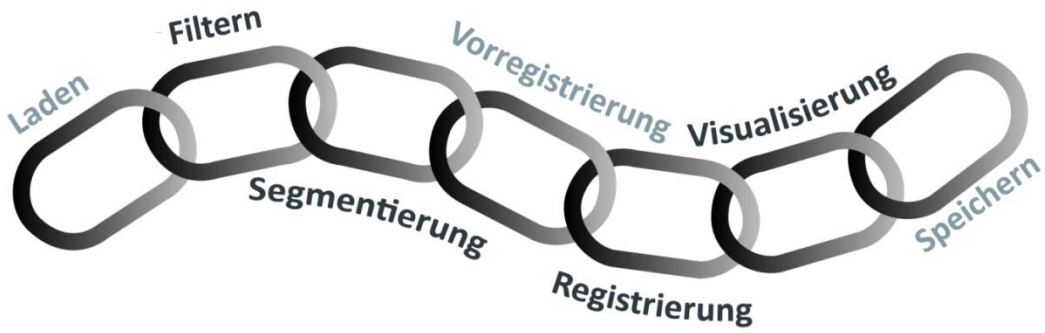


Abbildung 29: Prä- und Postverarbeitende Prozesse der Registrierung

Im Folgenden werden die verschiedenen Anforderungen an die Funktionseinheiten zusammengefasst.

Laden der Bilddaten

Um auf den Bilddatensätzen arbeiten zu können, müssen diese in den Hauptspeicher des Rechners geladen werden. Unter Anderem sollten die Dateiformate DICOM, RAW, Analyze, PNG bearbeitbar sein. Eventuell muss vorher eine Konvertierung in das MeVisLab interne Datenformat erfolgen (s. Kapitel 4.1.3). Ferner sollen Binärbilder bereits durchgeführter Segmentierungen in die Applikation geladen werden können.

Filterung der Bilddaten

Für die Bildvorverarbeitung sollen verschiedene Filter zur Verfügung stehen (s. dazu Kapitel 2.1.3). Dazu gehören beispielsweise ein Filter für das Beseitigen von Inhomogenitäten (Median- oder Gaußfilter) oder zur Kantenbetonung (Sobel Operator). Es soll möglich sein, verschiedene Filter nacheinander anwenden zu können.

Segmentierung der Strukturen

Es sollen verschiedene Segmentierungsverfahren zur Verfügung stehen. Dazu gehören ein einfaches Schwellwertverfahren (Threshold) und ein regionenbasiertes Verfahren (Region-Growing) (s. Kapitel 2.1.4.).

Vorregistrierung

Für die Optimierung oder zur Verringerung des zeitlichen Aufwands der später durchgeführten Registrierung, sollen hier vorbereitende Funktionen zur Verfügung stehen. Dazu gehören die Möglichkeit zum manuellen Vorausrichten der Datensätze sowie das Setzen oder Laden von Landmarken.

Registrierung

Dem Registrierungsalgorithmus müssen alle benötigten Daten zur Verfügung gestellt werden. Dazu gehören je nach Verfahren die Bilddatensätze, die segmentierten Strukturen und die gesetzten Landmarken. Fehlerhafte Parameter oder Ergebnisse müssen sichtbar angezeigt werden.

Visualisierung

Zur optischen Evaluierung oder einfach nur zur Sichtung der Daten werden verschiedene Bildviewer sowohl für den zweidimensionalen als auch dem dreidimensionalem Raum benötigt. Diese sollen sowohl Evaluierungen bezüglich der Lage der Strukturen als auch der Deckung der Bilddaten ermöglichen. Es sollte ein dreidimensionaler Viewer vorhanden sein, indem alle zur Verfügung stehenden Daten bei Bedarf dargestellt werden können.

Speichern

Die Anwendung muss das Speichern der unterschiedlichen Ergebnisdaten in verschiedenen Datenformaten wie DICOM, Analyze, etc. ermöglichen. Alle anderen, während des Prozesses gewonnenen Daten wie Transformationsmatrizen, gewählte Landmarken oder die Log-Dateien (im folgenden Absatz erläutert) müssen ebenfalls für den Anwender gespeichert werden.

Erweiterbarkeit

Neue Filtertechniken, Segmentierungs-, Registrierungs- wie Visualisierungsverfahren sollen einfach einzubinden und in die Gesamtanwendung integriert werden können. Einerseits um die neuentwickelten Verfahren testen und auswerten zu können, andererseits kann so die Anwendung für bestimmte Aufgaben optimiert werden, wie etwa für das automatisches Auswerten von Registrierungsergebnissen oder das Registrieren von multimodalen Aufnahmen. Dabei soll die Grundstruktur des Netzwerks nicht verändert werden.

Logging

Die Parameter aller verwendeten Filter-, Segmentierungs- und Registrierungsverfahren müssen aufgezeichnet und gespeichert werden. Für die Optimierung oder Evaluierung der angewandten Verfahren ist es nützlich, die bisher durchgeführten Schritte im Anschluss nachvollziehen und gegebenenfalls reproduzieren zu können.

5.2 Netzwerkkonzept

Bei der Netzwerkentwicklung müssen mehrere Eigenheiten der Datennetzwerke und des Datenflusses zwischen den Modulen beachtet werden. Im Vordergrund stand die Wahl der Netzwerkarchitektur oder -struktur.

5.2.1 Netzwerkstruktur

Für die Entwicklung des Netzwerks wurden drei Ansätze erörtert. Der erste war, in der durch die Registrierungsprozesskette festgelegten Reihenfolge, alle Module mit denen der nächsten Stufe zu verbinden (s. Abbildung 30). Dies wird natürlich mit steigender Anzahl an Modulen erstens unübersichtlich und zweitens ressourcenintensiv. Dies ist darauf zurückzuführen, dass jede Änderung eines Feldes alle daran gekoppelten Ereignisse auslöst. In diesem Fall führt eine Änderung des **Bild-Objekts** dazu, dass alle anschließenden Module im Netzwerk ihre Berechnungen auf Basis des Bild-Objekts neu durchführen [MeVF1] .

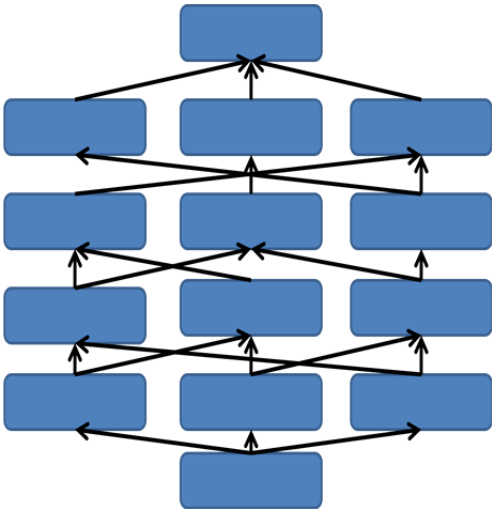


Abbildung 30: Erstes Strukturprinzip: Zwischen allen Modulen besteht eine Verbindung, jeweils von unten nach oben. Diese Struktur ist sehr ressourcenaufwendig.

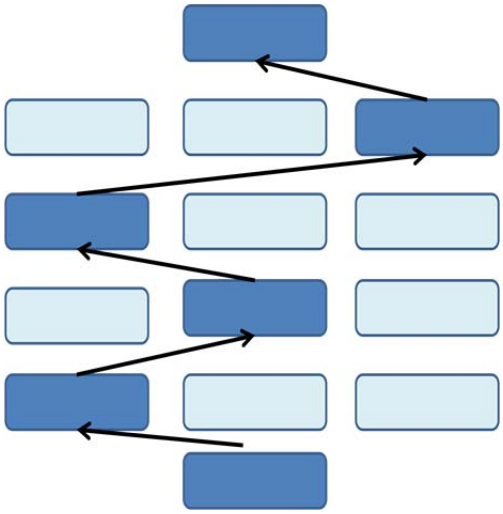


Abbildung 31: Zweites Strukturprinzip: Zwischen allen ausgewählten Modulen besteht eine Verbindung, jeweils von unten nach oben

Die zweite Möglichkeit bestand darin die Verbindungen zwischen den Modulen dynamisch hinzuzufügen, je nachdem welches Verfahren der Anwender auswählt. Dies entspräche weitestgehend einer geradlinigen Bildverarbeitungspipeline, dargestellt in Abbildung 31.

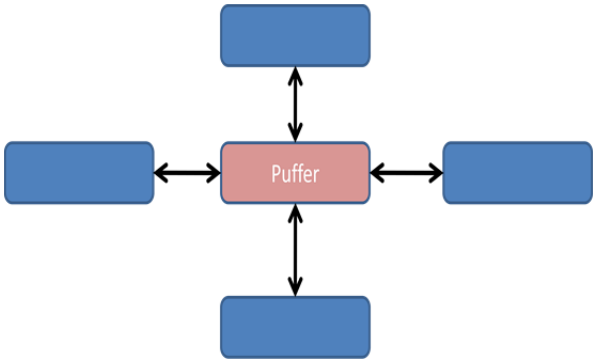


Abbildung 32: Drittes Strukturkonzept: Jeder verarbeitende Prozess greift auf ein, im Puffer gehaltenes Bild zu. Dies ermöglicht eine dynamische Bildbearbeitungsreihenfolge und erlaubt immer einen Zugriff auf die aktuellen Daten.

Die in Abbildung 32 dargestellte Variante macht sich das System des Puffers zunutze. Dabei wird nach jedem Bearbeitungsschritt das Bild in einem Puffer zwischengesichert. Dies hat den Vorteil dass die Reihenfolge der bildverarbeitenden Schritte nicht mehr vorgegeben sein muss. Für eine Bildverarbeitung ohne eine festgelegte Reihenfolge, erscheint Variante 3 als die geeignete Netzwerkstruktur.

Abbildung 33: Kombination der Strukturkonzepte zwei und drei: Das Bild wird weiterhin im Puffer gehalten, innerhalb der Makromodule (grün) wird dynamisch ein entsprechendes Modul zugeordnet.

Um die Komplexität und die Anzahl an neuen Verbindungen weiter zu verringern, wäre eine logische Schlussfolgerung, Module der gleichen verarbeitenden Gruppe in Makromodule zusammenzufassen und anschließend innerhalb des Makromoduls über einen Switch auf die verschiedenen Funktionen zuzugreifen. Dieses Konzept wird in Abbildung 33 dargestellt.
Der Vorteil dieser Struktur liegt darin, dass die Verbindungen und Funktionen innerhalb der Makromodule von denen des Gesamtnetzwerks abgekapselt sind und so das Gesamtnetzwerk **übersichtlicher** und **unkomplizierter** wird.
Ein weiterer Nutzen ist die **Erweiterbarkeit**. Innerhalb dieser Makromodule können jederzeit Module hinzugefügt oder geändert werden, ohne dass das übergeordnete Netzwerk verändert werden muss.

Die benötigten Modulgruppen fassen die Funktionalitäten in die folgenden Gruppen ein: **Laden**, **Puffer**, **Registrieren** (Vorregistrierung und Registrierung), **Filterung** und **Segmentierung**, **Visualisierung** und **Speichern**. Die Interaktionen zwischen den verschiedenen Modulgruppen werden in Abbildung 34 dargestellt.

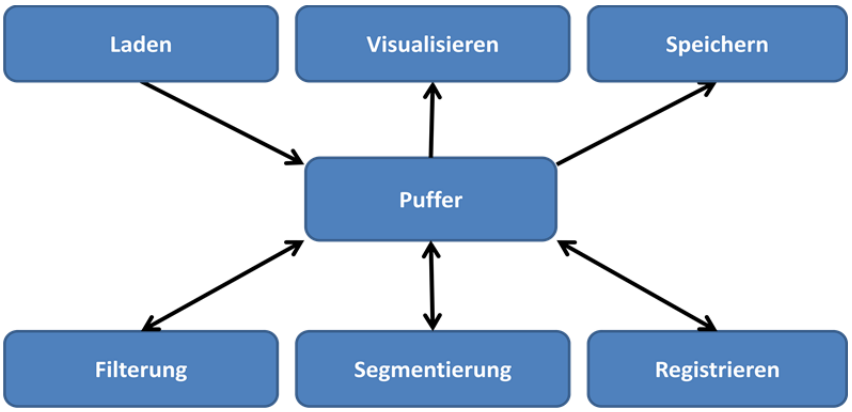


Abbildung 34: Makromodule in die Funktionalitäten der zu entwickelnden Applikation zusammengefasst sind.

Um die spätere Applikation Anwendern ohne entsprechende Lizenz zur Verfügung stellen zu können, darf sie die Maximalanzahl an neu entwickelten Modulen von 15 nicht überschreiten (s. Kapitel 4, [MevLic11]).

5.3 Konzept Benutzeroberflächenentwicklung

Das Design der Benutzeroberfläche orientiert sich an der Registrierungsprozesskette. Die komplette Anwendung wird in Tabs unterteilt, die jeweils einem bildverarbeitenden Schritt zugeordnet sind. Ferner wurde versucht die Kriterien des Papers „Qualitätskriterien für elektronische Publikationen in der Medizin“ [Schulz98] zum Thema Benutzeroberflächenentwicklung für medizinische Softwareprodukte einzuhalten.

Die Entwicklung der Benutzeroberfläche fand auf zwei Ebenen statt, der **Modulebene** und der **Applikationsebene**. Die in Abbildung 34 dargestellten Makromodule stellen auf der Modulebene sowohl die Funktionalitäten als auch Benutzeroberflächen der untergeordneten Module zur Verfügung.
Auf der Applikationsebene wurde ein Großteil dieser Benutzeroberflächen integriert (Abbildung 35, 3a und 3b), der Bestandteil an neuen Elementen auf der Applikationsebene ist relativ gering. Sie beschränken sich größtenteils auf die Menüführung (Abbildung 35, 1) und einer kleinen Übersicht über verschiedene Daten zur aktuellen Sitzung (Abbildung 35, 4).

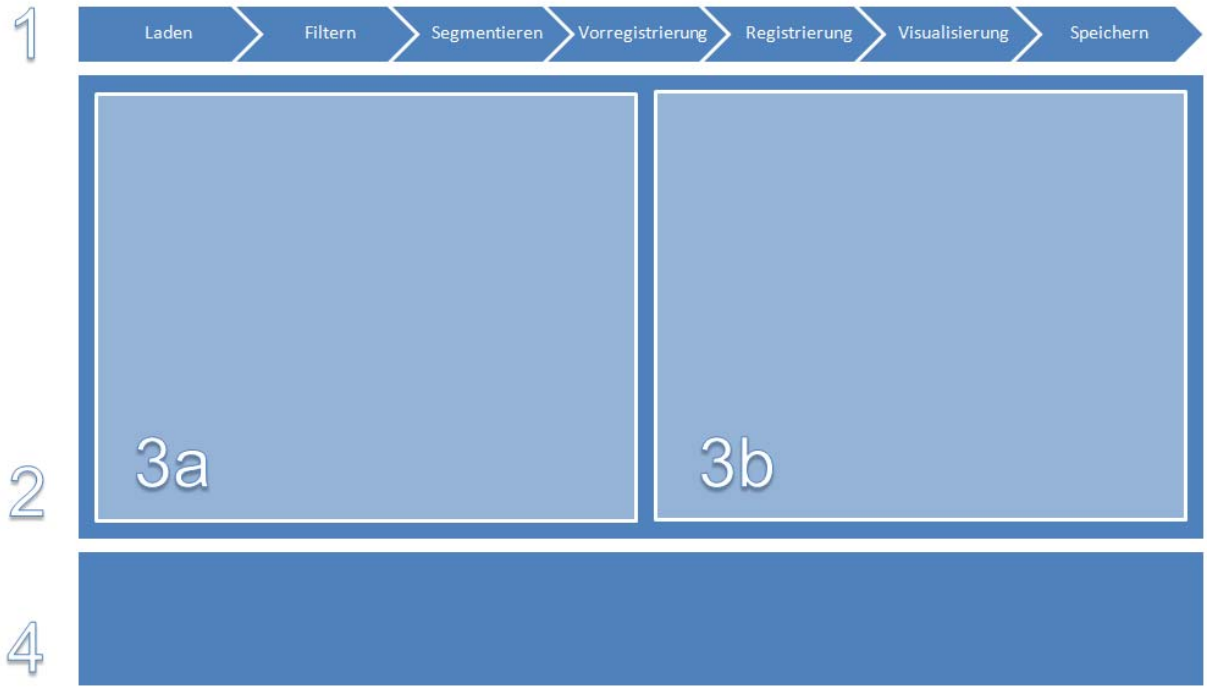


Abbildung 35: Entwurf der Benutzeroberfläche. (1) Menuführung (2) Bearbeitungsfenster (3a+b) Integrierte Moduloberflächen wie etwa Viewer, etc. (4) Overview

6 Entwicklung und Realisierung

Die Implementierung der Applikation vollzog sich in drei Phasen. Zuallererst stand die Entwicklung der benötigten Module. Daraufhin wurden sie dem Applikationsnetzwerk hinzugefügt und eine Benutzeroberfläche generiert. Anschließend wurden verschiedene Methoden entwickelt, die die Verbindungen zwischen den Modulen administrieren.

6.1 Modulentwicklung

Es wurden aus drei unterschiedlichen Gründen neue Module entwickelt. An erster Stelle stand natürlich die Entwicklung **neuer Funktionalitäten** (im Folgenden werden diese Module mit **f** gekennzeichnet). Dann wurden einige wenige vorhandene Module um kleinere Funktionalitäten **erweitert** (**e**). Als drittes wurden bestimmte Module in Makromodule **zusammengefasst** (**z**) um die innere Komplexität vom übrigen Netzwerk zu kapseln. Diese Kapselung hat zudem den Vorteil, dass innerhalb dieser Module Funktionalitäten geändert oder neu eingebaut werden können, ohne dass die äußere Netzwerkstruktur modifiziert werden müsste. Die entwickelten Module und deren Position in der Netzwerkhierarchie werden in Abbildung 36 aufgeführt.

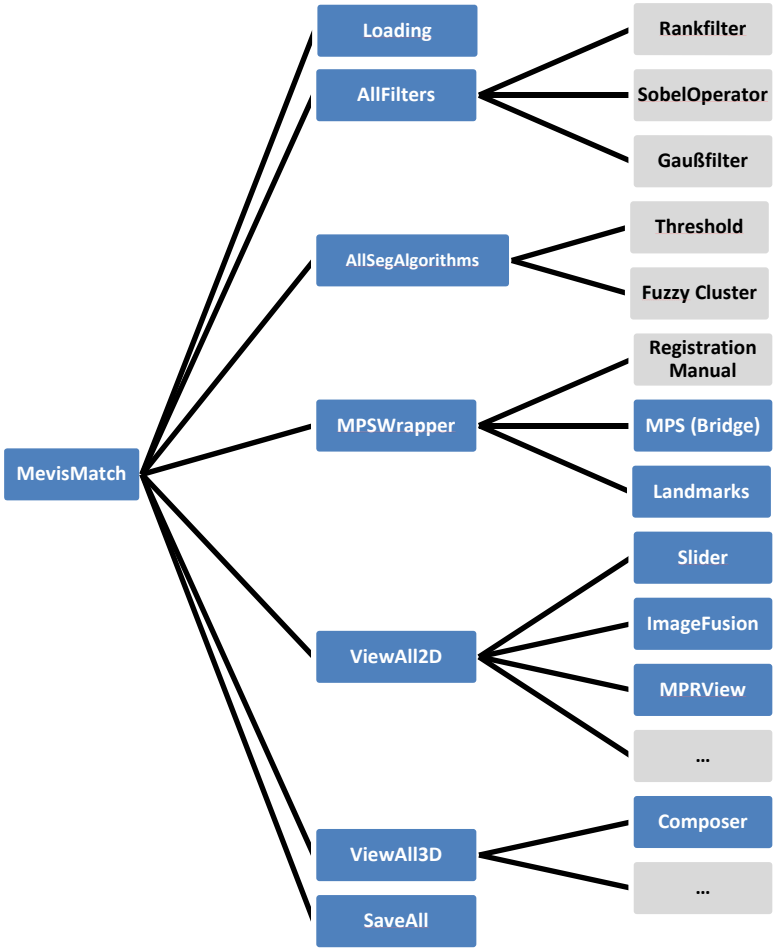


Abbildung 36: Vereinfachte Modulhierarchie im Netzwerk. Blau: Eigenentwicklungen, Grau: Vorhandene Module

Die entwickelten Module werden in die beiden Kategorien **Verarbeitende** und **Visualisierende Module** eingeteilt. Die Funktionen und der Aufbau der neuen Module werden in den folgenden beiden Kapiteln erläutert.

6.1.1 Verarbeitende Module

In diesem Kapitel werden alle Module zusammengefasst die manipulierend oder verwaltend auf die Bilddatensätze zugreifen. Sie werden in der Reihenfolge beschrieben, in der sie in Abbildung 36 aufgeführt werden.

Loading (e)
Bei diesem Modul handelt es sich um eine Erweiterung eines bereits vorhandenen Moduls zum Laden von Bildern. Wie in Kapitel 4.1.3 bereits erläutert, ist es nötig die zu verwendenden Bilddatensätze in das MeVisLab-interne Datenformat (DICOM/TIFF) umzuwandeln, bevor sie als dreidimensionale Datensätze in die Anwendung geladen werden können. Das Modul welches die Umwandlung durchführt liest allerdings nicht den korrekten Tag aus, der den Datensatz als Bildserie kennzeichnet (SeriesInstanceUID). Daher wird der Datensatz nicht als Serie anerkannt, sondern wird in einzelne Schichtbilder umgewandelt. Daher war es nötig ein Modul zu entwickeln, welches die konvertierten Bilddatensätze wieder zu einem dreidimensionalen Bild zusammenfügt.

AllFilters und AllSegAlgorithms (z)
Diese beiden Module kapseln jeweils Filteralgorithmen oder Segmentierungsverfahren und bieten eine einheitliche Schnittstelle nach außen sowie ein internes Logging.

Abbildung 37: **Modulstruktur des Moduls AllFilters.**
Viewer1 stellt das Ausgangsbild dar, Viewer2 das aktuell im internen Puffer befindliche Bild, Viewer3 das Ergebnis von Filter4 auf das im Puffer befindliche Bild.
Das Modul AllSegAlgorithms ist analog aufgebaut.

Beide Module sind nach dem in Abbildung 37 links dargestellten Schema aufgebaut. Dabei wird das in das Modul eingehende Bild zunächst in den internen Puffer geschrieben (zu sehen in Viewer 1). Dieser reicht das Bild über einen Switch an den vom Anwender ausgewählten Algorithmus (in diesem Fall Filter 4). Ist der Anwender mit dem Ergebnis zufrieden, wird das bearbeitete Bild dem Puffer wieder zugeführt und das alte Bild im Puffer wird überschrieben. Dieses Verfahren hat den Vorteil dass mehrere Filter nacheinander angewendet werden können und der Ausgangsdatensatz trotzdem erhalten bleibt. Erst nach Abschluss der Bearbeitung wird der Ausgangsdatensatz mit dem bearbeiteten überschrieben. Dies erfolgt außerhalb de Moduls.

Die Verfahren die die beiden Module zur Verfügung stellen sind in Abbildung 36 aufgeführt und wurden bereits in Kapitel 2.1.3 und Kapitel 2.1.4 erläutert.

MPSWrapper (z)
Innerhalb des MPSWrapper Moduls werden die Bridge und einige, die Transformation betreffende, Module zusammengefasst. Nach der Registrierung stellt das Modul das transformierte Objektbild, samt entsprechendem Binärbild sowie die Transformationsmatrix zur Verfügung. Neben der Transformation ermöglicht der MPSWrapper das Loggen der Registrierungsparameter wie etwa den ausgewählten Registrierungsalgorithmus, die verwendeten Daten oder während des Registrierungsprozesses aufgetretene Fehlermeldungen.

MPS (Bridge)
Dieses Modul ist keine Eigenentwicklung, es handelt sich dabei um die in Kapitel 2.2.5.1 beschriebene Bridge zwischen MeVisLab und den Algorithmen der Bibliothek MatchPoint (s. Kapitel 2.2.5). Sie reicht die eingehenden Daten (Bilder, Landmarken und segmentierte Strukturen) an einen passenden MatchPoint-Algorithmus weiter und erhält aus diesem die Transformationsmatrix. Sie unterstützt bisher die affine Registrierung.

Landmarks (f)
Ein Hauptproblem bei dem Setzen von Landmarken ist das der Anwender oft die Reihenfolge der gesetzten Marker durcheinanderbringt. Daher wurde das Modul Landmarks entwickelt, welches vom Anwender das paarweise Setzen von Landmarken fordert.
Dazu wurden zwei Module zum Setzen von Markern aneinander gekoppelt. Wird mit einem Modul ein Marker gesetzt, wird es deaktiviert bis im anderen Bild ein Marker gesetzt wird. Zusätzlich zu dieser festgelegten Markierreihenfolge werden die Landmarkenpaare immer in den gleichen Farben dargestellt. Ferner ist die Möglichkeit gegeben, zu jedem gesetzten Marker über einen eigenen Button zu navigieren.
Aus den gesetzten Landmarken werden zeitgleich einige Parameter berechnet, wie etwa der Abstand zwischen den Markerpaaren oder die Varianz und Standardabweichung der Koordinaten. Diese Parameter können für eine spätere Evaluierung der Registrierung herangezogen werden, der durchschnittliche Abstand zwischen den Markerpaaren wäre eine geeignete Kennziffer.
Diese Berechnung wird mit Python durchgeführt, anhand der Abstandsformel Formel 2.

Formel 2: Euklidische Abstandsformel

$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2}$$

Denkbar wäre auch eine Implementierung als ML-Modul gewesen, worauf auf Grund der bereits fortgeschrittenen Modulanzahl verzichtet wurde.

Es wäre wünschenswert gewesen, wenn MeVisLab ein ähnliches Pufferprinzip wie bei den Bildern für die Markerobjekte bereitstellen würde. Dadurch wäre es möglich gewesen, alle Daten an einem zentralen Ort zu puffern. Da ein solcher Puffer nicht zur Verfügung steht, verbleiben die gesetzten Marker im Landmarks-Modul und werden auch in diesem weiter verarbeitet. Auch die Transformation der Marker wird innerhalb dieses Moduls durchgeführt.

SaveAll (z)
Dieses Modul erlaubt das Speichern aller während des Bildverarbeitungsprozesses gewonnen Daten. Der Anwender hat die Möglichkeit bestimmte Bilddatensätze auszuwählen und sie u.A. in den Formaten DICOM, Analyze oder PNG abzuspeichern. Ferner können gesetzte Marker im XML-Format abgespeichert werden. Zudem werden alle in dieser Sitzung angefallenen Log-Dateien (s. Kapitel 6.1.3) aus ihrem temporären Verzeichnis in das vom Anwender angegebene Verzeichnis kopiert, sodass sie den abgespeicherten Bilddatensätzen direkt zugeordnet werden können.

6.1.2 Visualisierende Module

Zu den visualisierenden Modulen werden alle im Rahmen dieser Arbeit neu entwickelten Module gezählt, die die vorhandenen Bilddaten in einer für den Anwender aussagekräftigen Form für die Evaluierung der Registrierungsergebnisse zur Verfügung stellen.
Wie in Kapitel 2.1.5.2 beschrieben wird die Transformation nicht auf die Bilddaten angewandt sondern auf dessen Weltkoordinaten. Daher sind nur solche Viewer für die Registrierung aussagekräftig, die in irgendeiner Form mit Overlays (Bildüberlagerung) arbeiten oder auf andere Weise das Lageverhältnis beider Bilddaten auf einmal anzeigen.

Slider (f)
Bei dem Slider Modul handelt es sich um eine Eigenimplementierung, bei dem Objekt- und Referenzbild übereinander liegen und das Referenzbild „aufgedeckt“ wird. Dieser Viewer erlaubt es, Bildkomponenten an diesen Aufdeckungsgrenzen eingehend zu vergleichen.
Das Prinzip wird mittels eines Overlays des Objektbildes über das Referenzbild realisiert. Dieses wird je nach Mausposition und ausgewähltem Status zugeschnitten. Zu den verschiedenen Status gehören eine vertikale, eine horizontale und eine freie Box-Ansicht (Seethrough), dargestellt in Abbildung 38.

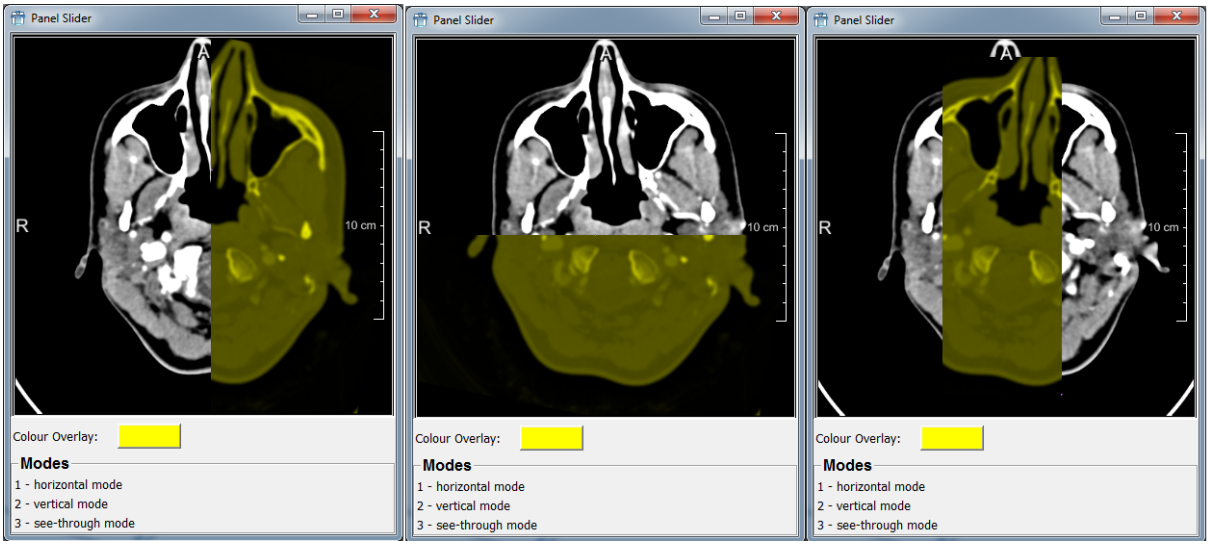


Abbildung 38: Das Slider-Modul mit den drei verschiedenen Stati: Vertical (links), Horizontal (mittig) und See-through (rechts)

ImageFusion (f)
Bei der ImageFusion (engl. für Bildfusion) werden die Grauwerte von sowohl Objekt- als auch Referenzbild auf eine Farbe transformiert und anschließend mit Hilfe eines Overlays übereinander gelegt. Bei der Farbwahl wird idealerweise die Farblehre beachtet, bei denen zwei Farben gemischt eine dritte ergeben (standardmäßig Grün und Rot, ergeben Gelb). Jene Grauwertbereiche die sich hinsichtlich der Grauwertintensität gut gleichen, erscheinen in der Mischfarbe. Nicht deckungsgleiche Bereiche erscheinen weiterhin in den Ausgangsfarben.
Diese Visualisierungstechnik wird oft zur Darstellung von Registrierungsergebnissen verwendet, war aber in dieser Form noch nicht in MeVisLab implementiert.

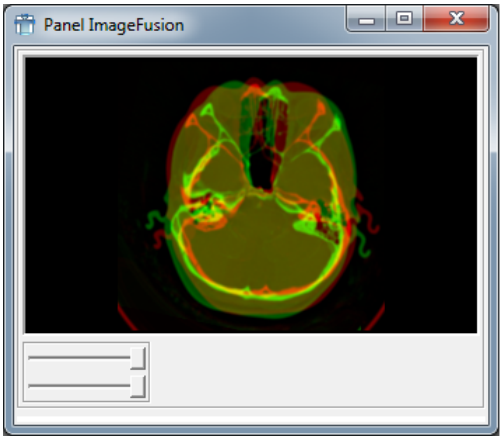


Abbildung 39: Darstellung von zwei Schädel CTA- Aufnahmen mit dem Modul Image Fusion

MPRView (e)

Dieses Modul erweitert die Möglichkeiten des MPR-Moduls um einige Sichten (s.Kapitel 2.1.2). Dazu definiert der Anwender über eine frei bewegliche Ebene die gewünschte MPR-Schicht. Diese wird in einem zweidimensionalen Viewer dargestellt (Abbildung 40, links). Zusätzlich beinhaltet das Modul noch dreidimensionale Viewer in denen die MPR-Schicht im Verhältnis zum volumengerenderten Bild oder zur segmentierten Struktur dargestellt wird (Abbildung 40, mittig und rechts).

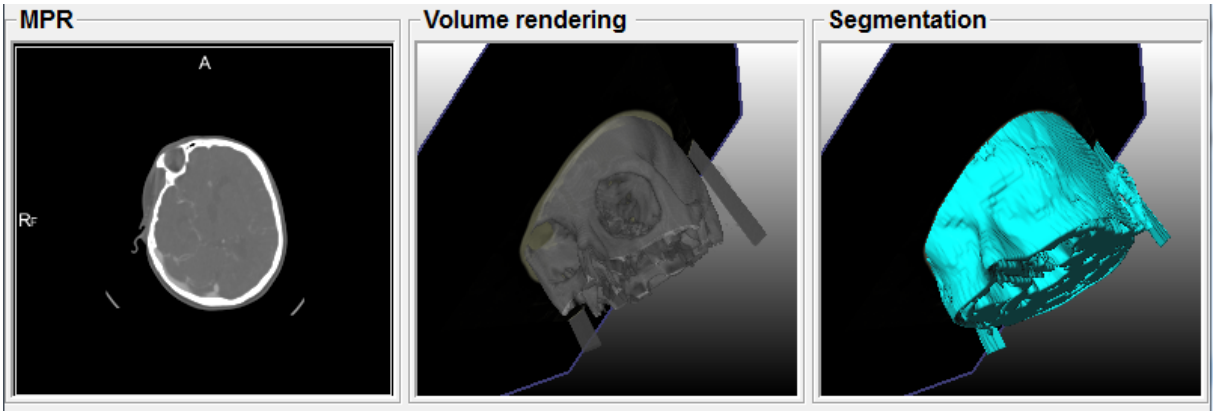


Abbildung 40: Bestandteile des MPRView-Moduls. Links: MPR Mitte: Volume Rendering des Datensatzes Rechts: Darstellung der segmentierten Strukturen über Oberflächenvisualisierung

ViewAll2D und ViewAll3D (z)

Die Module **ViewAll2D** und **ViewAll3D** fassen mehrere Viewer zusammen, die über eine einheitliche Schnittstelle angesprochen und über eine gemeinsame Benutzeroberfläche gesichtet werden können.

Die Viewer wurden in die Gruppen zweidimensional und dreidimensional getrennt, da die internen Beipässe unterschiedlich angesprochen und reguliert werden müssen. Wie in Kapitel 5.2.1 bereits erläutert, führt ein stark verzweigtes Netzwerk zu einem hohen Rechenaufwand. Dies fällt insbesondere bei Modulen für die dreidimensionale Visualisierung auf.-

Daher wurden für die zweidimensionalen und die dreidimensionalen Bildviewer eigene Makromodule entwickelt, deren Moduleingänge einzeln deaktiviert werden können. Das Prinzip ist in Abbildung 41 dargestellt. Durch diese gezielte Datenweitergabe kann der Rechenaufwand beim Ansprechen des Moduls minimiert werden.

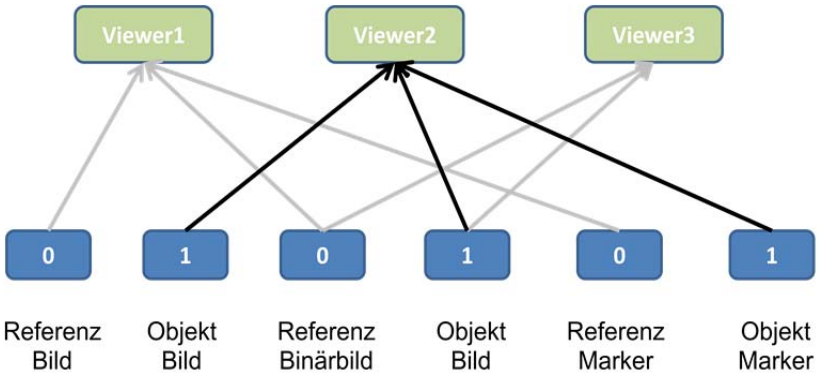


Abbildung 41: Beispielhafte Vernetzung innerhalb der ViewAll2D und ViewAll3D Module. Je nach Angaben des Anwenders oder des ausgewählten Viewers werden bestimmte Bypass-Module aktiviert bzw. deaktiviert. Dadurch können bestehende Verbindungen innerhalb der Module beibehalten und trotzdem die Zugänge zu den einzelnen Viewermodulen kontrolliert werden.

In der untenstehenden Tabelle 5 ist zu erkennen, dass der Speicherbedarf bei dauerhaft aktiven Moduleingängen sofort stark ansteigt. Werden die Eingänge aber viewer-abhängig aktiviert, ist der Speicherbedarf entsprechend geringer bis zu dem Zeitpunkt, an dem alle Viewer vom Anwender betrachtet wurden.

Eingänge	Speicherbedarf
Viewer ohne Beipass, geschlossen	15879 KB
Viewer geschlossen	344 KB
Viewer geöffnet, erste Ansicht aktiv	3073 KB
Viewer geöffnet, alle Ansichten aktiv	15879 KB

Tabelle 5: Speicherbedarf des Moduls ViewAll3D mit und ohne aktiviertem Bypasssystem

Im **ViewAll2D** wurden mehrere Viewer zusammengefasst, unter Anderem ein Viewer für die drei Standardsichten mit der Möglichkeit die segmentierten Strukturen per Overlay darzustellen. Überdies beinhaltet das ViewAll2D die Module **Slider**, **ImageFusion** und **MPRView.**

Das **ViewAll3D**-Modul bietet für sowohl Referenz- als auch Objektbild einen dreidimensionalen Viewer, bei dem bei Bedarf die segmentierten Strukturen mit eingeblendet werden können. Die beiden segmentierten Strukturen werden außerdem in einem eigenen dreidimensionalen Viewer dargestellt. Darüber hinaus beinhaltet das ViewAll3D-Modul den Gesamtviewer, der in die Anforderungsanalyse in Kapitel 5.1 aufgenommen wurde. Er wird in der Benutzeroberfläche als **Composer** aufgeführt.

Laut Anforderungsanalyse hätte dieser Gesamtviewer für die gesamte dreidimensionale Bilddarstellung ausgereicht. Allerdings ist aus unbekannten Gründen der Modultyp, auf dem dieser Viewer arbeitet, nicht so stabil wie die der kleineren, spezialisierten Viewer. Dies zeichnet sich durch Flackern oder langwierige Bildberechnungen aus. Daher wurde, entschieden die spezialisierten Viewer zusätzlich zum Multifunktionsviewer mit aufzunehmen.

6.1.3 Logging

Wie im vorherigen Kapitel erwähnt müssen bei allen bildverändernden Maßnahmen die entsprechenden Parameter geloggt werden. Dies gilt sowohl für die Filterung, die Segmentierung als auch für die Registrierung, da in diesen Bereichen manipulierend auf die Bilder zugegriffen wird. MeVisLab bietet keine solche Logging-Funktion. Es können zwar Nachrichten auf der Konsole ausgegeben und diese anschließend gespeichert werden, allerdings gibt es keine Möglichkeit die eigenen Nachrichten von denen des Systems oder fremden Modulen zu trennen. Allerdings existiert ein Modul namens **SettingsManager**. Dieses Modul wird dazu gebraucht Parameterfelder auszulesen und deren Werte abzuspeichern. Geringfügig modifiziert, kann dieses Modul zum Logging verwendet werden, dabei werden nach jeder applizierten Änderung der Bilddaten alle Parameter gespeichert.

Sobald der Anwender signalisiert, dass eine Änderung dauerhaft auf das Bild appliziert werden soll, werden die aktuellen Modulparameter in einer Log-Datei innerhalb eines temporären Verzeichnisses gespeichert. Diese beinhaltet zusätzlich den Namen des Bildes und das Originalverzeichnis. Für eine eindeutige Zuordnung von Log-Dateien zu einer Sitzung wird mit Hilfe von Python eine 16-stellige eindeutige ID generiert. Diese Session-ID wird jedem loggenden und speichernden Modul übergeben, welche der Log-Datei hinzugefügt wird. So kann jede Log-Datei eindeutig einem Bilddatensatz, einer Sitzung und einem Zeitpunkt zugeordnet werden. Die Log-Dateien werden mit einer entsprechenden Zeitmarke und der Session-ID versehen.

6.1.4 Netzwerk- und Benutzeroberflächenentwicklung

In Abbildung 42 ist das zusammengestellte Netzwerk dargestellt. Im Ausgangszustand sind nur einige Verbindungen festgelegt, der Großteil wird dynamisch hinzugefügt, wie später in Kapitel 6.2 erläutert wird.

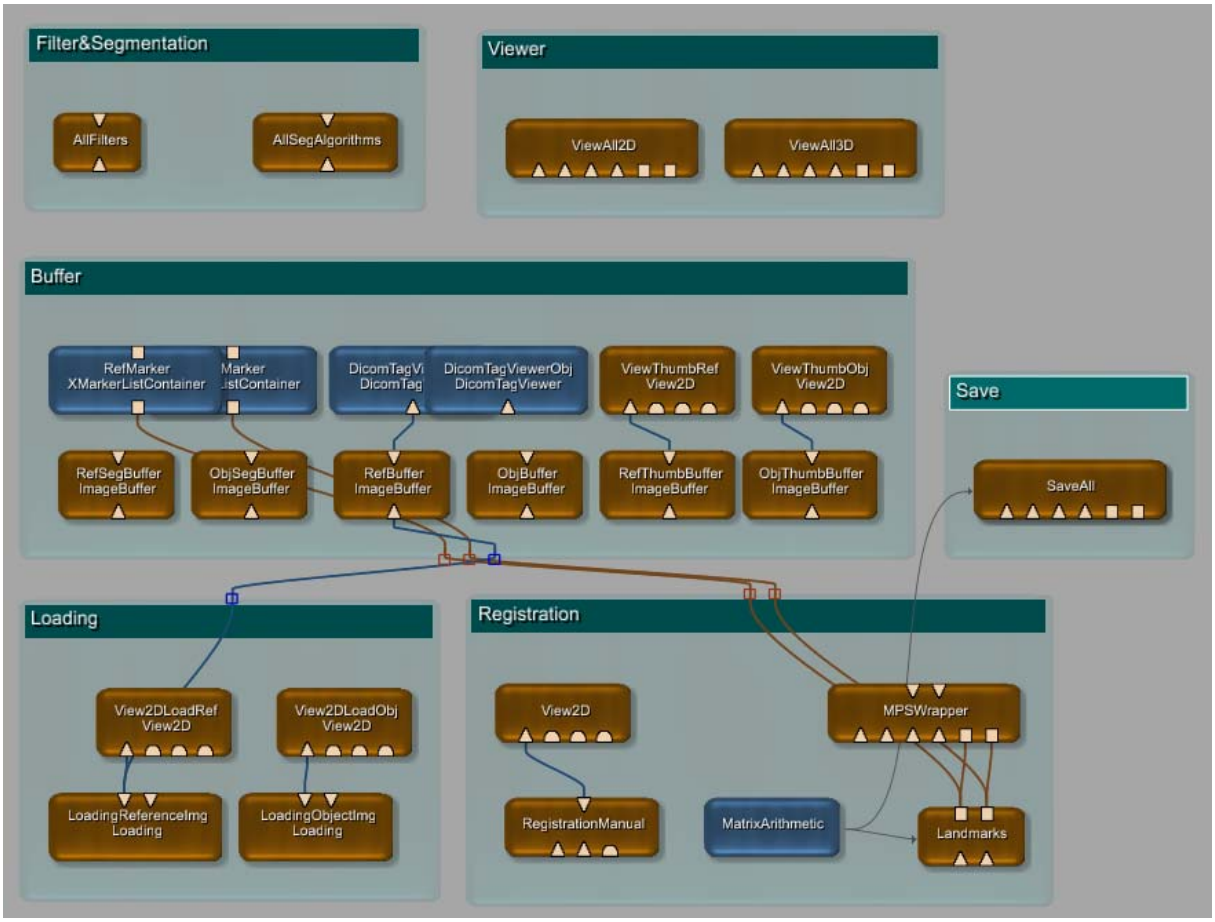


Abbildung 42: Netzwerk der Anwendung MevisMatch im Startzustand

Die Benutzeroberfläche wurde der in Kapitel 5 konzipierten nachempfunden, sie ist in Abbildung 43 dargestellt.

Abbildung 43: Endgültige Version der Benutzeroberfläche.

6.2 Verbindungen und weitere Netzwerkfunktionalitäten

Der korrekte Datenfluss zwischen den Modulen ist von entscheidender Bedeutung für die Funktionalität und ist entsprechend wichtig für Lauffähigkeit und Stabilität der Anwendung.

Wie im vorherigen Kapitel beschrieben, sind an die verschiedenen Tabs des Menüs bestimmte Methoden gekoppelt, die die Verbindungen je nach Anforderung neu setzen. Dies ist in der Abbildung 44 gut zu erkennen.

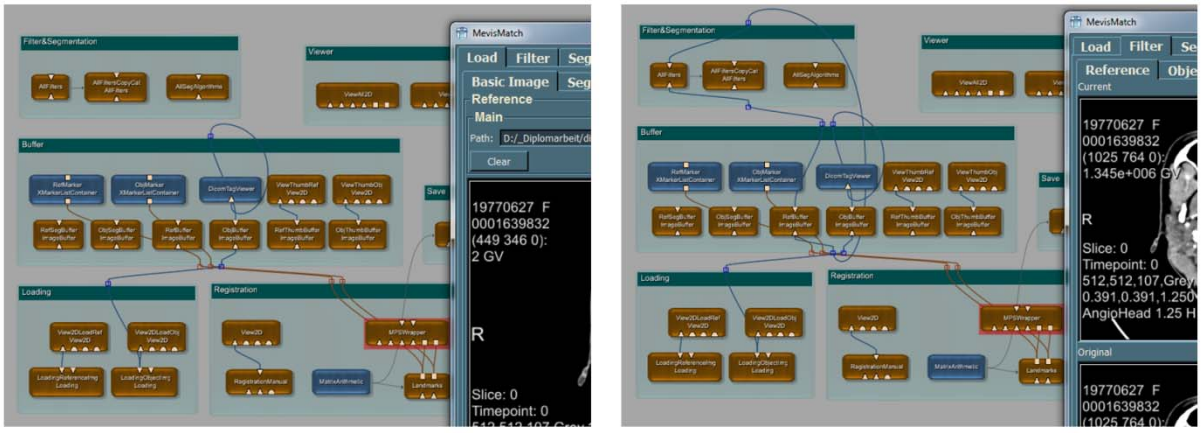


Abbildung 44: links - Netzwerkstatus im Tab Laden rechts - Netzwerkstatus im Tab Filtern

Zum Beispiel werden im Anschluss an jede Bilddatenänderung (Laden, Filtern, Segmentieren, Registrieren) die verarbeitenden Module mit dem entsprechenden Puffer verbunden, der Ausgangsdatensatz überschrieben und die Verbindung wieder abgebaut, um den Netzwerkrumpf simpel zu halten.

Um den Datenfluss dabei stabil und fehlerfrei zu halten ist beim Arbeiten mit MeVisLab das **Timing** ausschlaggebend. Die durch die FieldListener ausgelösten Ereignisse (s. Kapitel 4.1.1) werden unkoordiniert und nach unterschiedlichen Prioritäten ausgelöst. Feldänderungen in **ML-Modulen** lösen sofort an das Feld gekoppelte Ereignisse und Berechnungen aus. Feldänderungen **von Open Inventor-Modulen** hingegen, werden zunächst in einen Stack geschrieben und es gibt keine Regeln wann genau sie ausgeführt werden. (s. [MeVisTut10] , Kapitel 7.1). **Skriptbefehle** werden sequentiell abgearbeitet, ohne Rücksicht darauf ob der vorhergehende Befehl bis zum Ende durchgeführt wurde. Dies hat einerseits den Vorteil dass mehrere Berechnungen parallel gestartet werden können, andererseits aber den schwerwiegenden Nachteil das bestimmte Befehlsfolgen (wie etwa das dynamische Verbinden und Speichern in einem Puffer) nicht im Sinne des Entwicklers durchgeführt werden. Beispielsweise wurden die Verbindungen zum Puffer gekappt, ohne dass eine vollständige Datenübertragung stattgefunden hat (s. Abbildung 45). Beim Entwickeln einer großen Anwendung muss daher auf diese drei Aspekte Rücksicht genommen werden.

Dazu werden in diesem Falle bestimmte Befehlssequenzen unterbrochen, bis die voran gegangenen Befehle abgearbeitet sind. Dies ist in Abbildung 45 dargestellt.

Abbildung 45: Gegenüberstellung der Zeitachsen und Ergebnisse des Schreibens in einen Puffer – mit und ohne Pausieren. (a) Verbinde zum Puffer (b) Schreibe in den Puffer (c) Löse Verbindung zum Puffer (p) Warte bis alle anstehenden Prozesse fertig sind

Ein akutes Problem ist das Netzwerke und Module weder mit Standardparametern initialisiert noch resettet werden können. Wird in einem Modul oder Netzwerk ein Parameter verändert, wird er beibehalten, unabhängig davon ob der Entwickler es explizit abspeichert oder nicht. Dies mag bei kleinen Netzwerken als Feature angesehen werden, bei Anwendungen hingegen ist es eher unpraktisch.

Dementsprechend müssen zum Initialisieren bzw. Resetten des Netzwerks von MevisMatch alle wichtigen Parameter (wie Transformationsmatrix, etc.) und Verbindungen manuell neu gesetzt werden.
Durch die Implementierung dieser Initialisierungsmethoden löst sich zudem ein weiteres Problem. Dadurch dass der Anwender MevisMatch aus MeVisLab heraus starten muss, ist die Gefahr gegeben, dass er nach Beenden der Anwendung ein modifiziertes Netzwerk aus Versehen abspeichert. Durch die Initialisierungsmethoden können zumindest kleinere Modifikationen wieder ausgeglichen werden.

6.3 Dokumentation

Zur Dokumentation zählen vier verschiedene Medien: die **Beispielnetzwerke**, die **Hilfedateien** der Module, die **Dokumentation der Skript-** und **Pythondateien** sowie ein **Handbuch**. Auf alle Dokumentationen (bis auf das Handbuch) kann innerhalb von MeVisLab über die Menüs der Module zugegriffen werden.

Zu jedem entwickelten Modul wurde ein Beispielnetzwerk erstellt. Der Anwender kann anhand dieser Netzwerke auf die Funktionsweise und auf die für das Modul benötigte Netzwerkstruktur zugreifen. Zu jedem Modul wurde ferner eine Hilfedatei erstellt. Darin werden alle wichtigen Parameter sowie das Ansprechen der Module per Skript erläutert. Die wichtigsten Python-Methoden wurden in Englisch kommentiert. Damit die Einstiegspunkte für spätere Modifikationen am Quellcode einfach zu identifizieren sind wurden sie entsprechend kommentiert. Das Handbuch steht in Form eines PDFs zur Verfügung. Darin werden die Installation, die Anwendung und das Erweiterungsprinzip von MevisMatch erläutert.

7 Ergebnisse

Im Folgenden werden die Ergebnisse der vorliegenden Arbeit erörtert. Dazu wird zunächst ein typischer Arbeitsablauf mit MevisMatch geschildert. Anschließend werden verschiedene Aspekte der Anwendung bewertet, wie die Benutzeroberfläche, die Erweiterbarkeit oder die Qualität der Anwendung.

7.1 Anwendung der Applikation MevisMatch

In diesem Beispiel werden zwei CTAs (computertomographische Angiographie) eines Schädels mit jeweils 56 MB Größe und jeweils 128 Schichten verwendet, s. Abbildung 46.

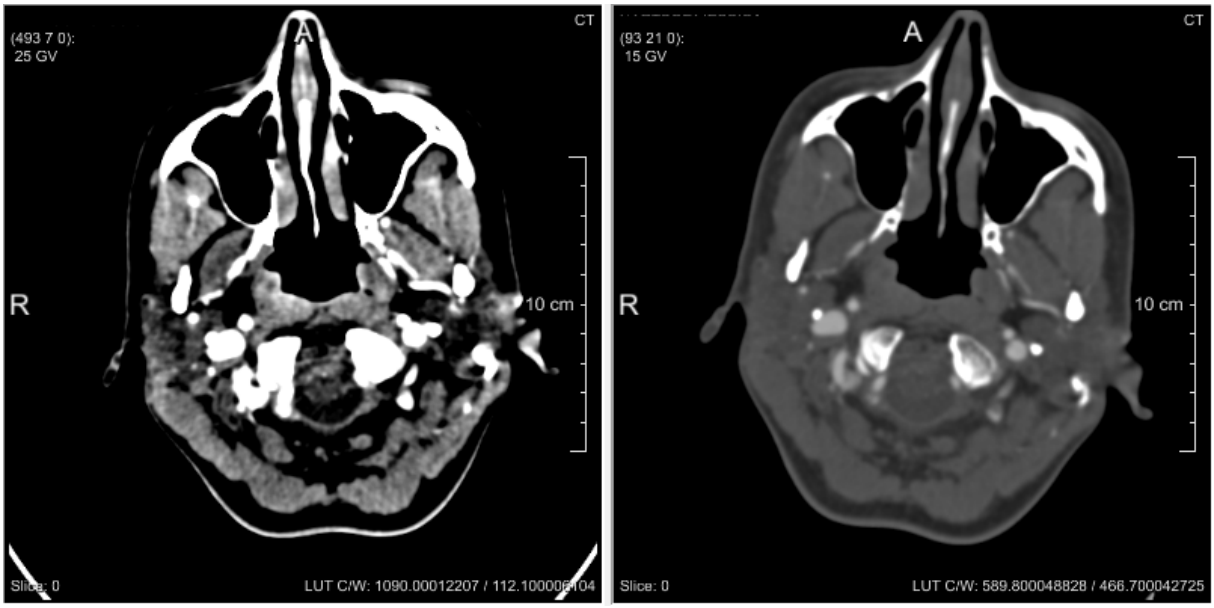


Abbildung 46: Ausgangsdatensätze für den Beispieldurchlauf. Links: Referenzbild, Rechts: Objektbild

7.1.1 Laden der Bilddatensätze

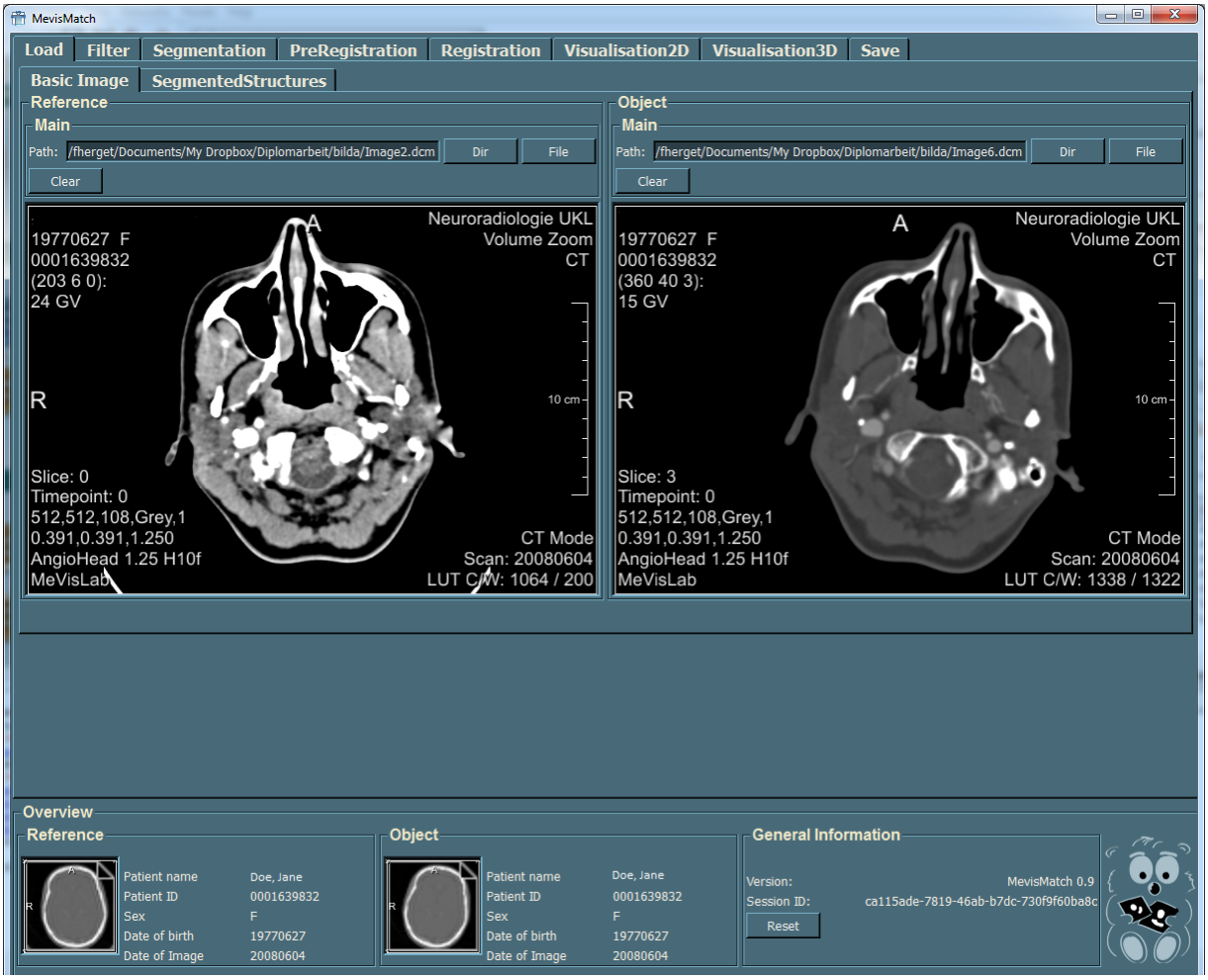


Abbildung 47: Tab zum Laden der Bilddatensätze

Zunächst werden die Ausgangsdatensätze in die Anwendung geladen. Wie in Abbildung 47 zu sehen wird auf der linken Seite das Referenz-Bild, auf der rechten Seite das Objekt-Bild dargestellt. Nach dem Laden der Bilder sind verschiedene Metainformationen zu dem ausgewählten Bild verfügbar, siehe Abbildung 48. Falls für diesen Datensatz bereits segmentierte Bilder bestehen, können diese im Tab **Segmented Structures** nachgeladen werden.

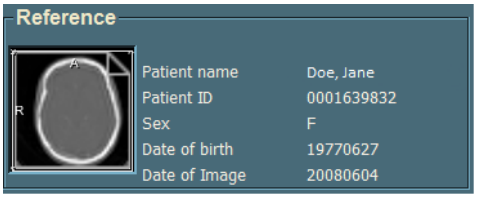


Abbildung 48: Metadaten des Testdatensatzes aus dem Feld "Overview"

Abbildung 49: Über den Tab "Segmented Structures" können bereits vorhandene Binärbilder für segmentierte Strukturen geladen werden.

Wenn auf Basis der vorher geladenen Bilddaten segmentiert werden soll, muss über die Funktion **Copy from File** der entsprechende Datensatz kopiert werden. In diesem Fall soll neu segmentiert werden, daher wird die **Copy from File**- Funktion genutzt (s. Abbildung 49).

Es bestehen nun vier Bilddatensätze: Referenz- und Objektbild, sowie von jedem der beiden eine Kopie, auf der später die Segmentierung durchgeführt wird. Diese werden im Folgenden als **Segmentierungskopien** bezeichnet.

7.1.2 Filtern

Abbildung 50: Tab zum Filtern der Segmentierungskopie des Referenzbildes. Hier wurde ein Sobel-Operator angewandt, aber noch nicht auf den Netzwerkpuffer übertragen, s. Darstellung des Ausgangsdatensatzes rechts unten.

Im nächsten Fenster **Filter** können die Bilddatensätze mit Hilfe verschiedener Filter bearbeitet werden. In diesem Fall wird auf allen vier Bilddatensätzen ein Medianfilter angewandt, um Rauschen zu entfernen und ohne dass die Schärfe der Kanten verloren geht. Anschließend wird zur Vorbereitung der Segmentierung auf den Segmentierungskopien ein Sobel Operator angewandt (s. Abbildung 50). Näheres zu Aufgaben und Funktionsweise der angewandten Filter sind Kapitel 2 zu entnehmen.

7.1.3 Segmentieren

Abbildung 51: Tab zum Segmentieren. Hier wurde das Region-Growing Verfahren angewandt um die knöchernen Strukturen des Schädels zu segmentieren.

Im Tab **Segmentation** können verschiedene Segmentierungsverfahren angewandt werden (s. Abbildung 51). In diesem Falle wird auf beide Bilder ein Region-Growing Verfahren angewendet (s. Kapitel 2.1.4), da sich die knöchernen Strukturen des Schädels gut vom Rest abheben. Die Segmentierung des Referenzbildes ist in Abbildung 52 links oben dargestellt, die des Objektbildes links unten. Die rechte Darstellung beinhaltet beide segmentierte Strukturen. Aus ihr lässt sich erkennen, dass Referenz- und Objektbild annähernd die gleiche Skalierung und Position im Raum besitzen. Das Objektbild ist im Verhältnis ein wenig länger und um die Z-Achse rotiert. Nach Deutung dieser Lageverhältnisse können später bestimmte Registrierungsalgorithmen ausgewählt werden.

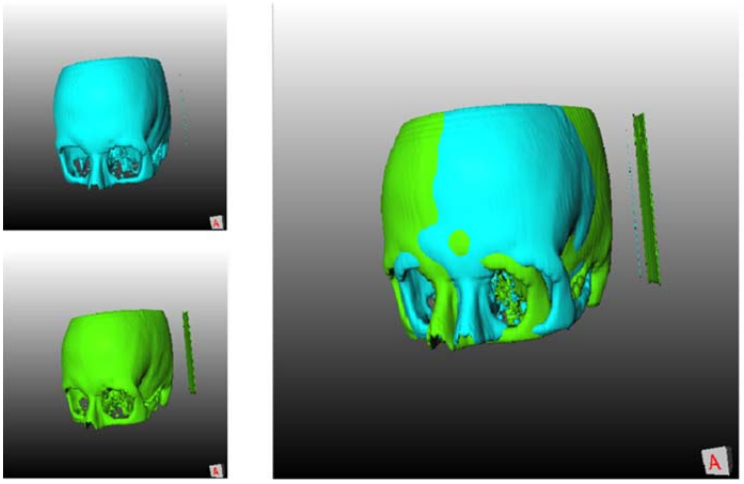


Abbildung 52: Segmentierte Strukturen:
LO - Referenzstruktur,
LU - Objektstruktur,
R - Beide Strukturen im Lageverhältnis zueinander

7.1.4 Vorbereitung für die Registrierung

Abbildung 53: Tab für die Registrierungsvorbereitung. Unter PreRegistration kann der Objektdatensatz manuell auf den Referenzdatensatz ausgerichtet werden.

Wie in Abbildung 53 zu sehen, beinhaltet das Tab **PreRegistration** zwei für die Registrierungsvorbereitung wichtige Funktionen, das **manuelle Registrieren** (Manual Positioning) und das **Setzen von Landmarken** (Landmarks).
Unter dem **Manual Positioning** ist es möglich die beiden Ausgangsdatensätze manuell aufeinander auszurichten. Für viele Registrierungsalgorithmen ist es notwendig oder zeitsparend wenn die Bilddaten grob überlappen oder aufeinander ausgerichtet sind. In diesem Fall wird das Objektbild leicht rotiert und verschoben (Translation). Das Ergebnis ist in Abbildung 54 dargestellt.

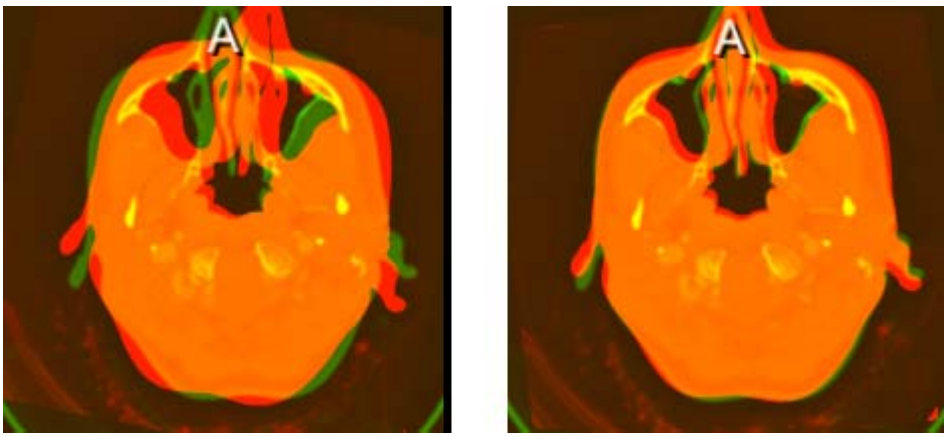


Abbildung 54: Links - Ausgangslage der beiden Bilder zueinander, Rechts - Manuell registrierte Lage der Bilder zueinander.

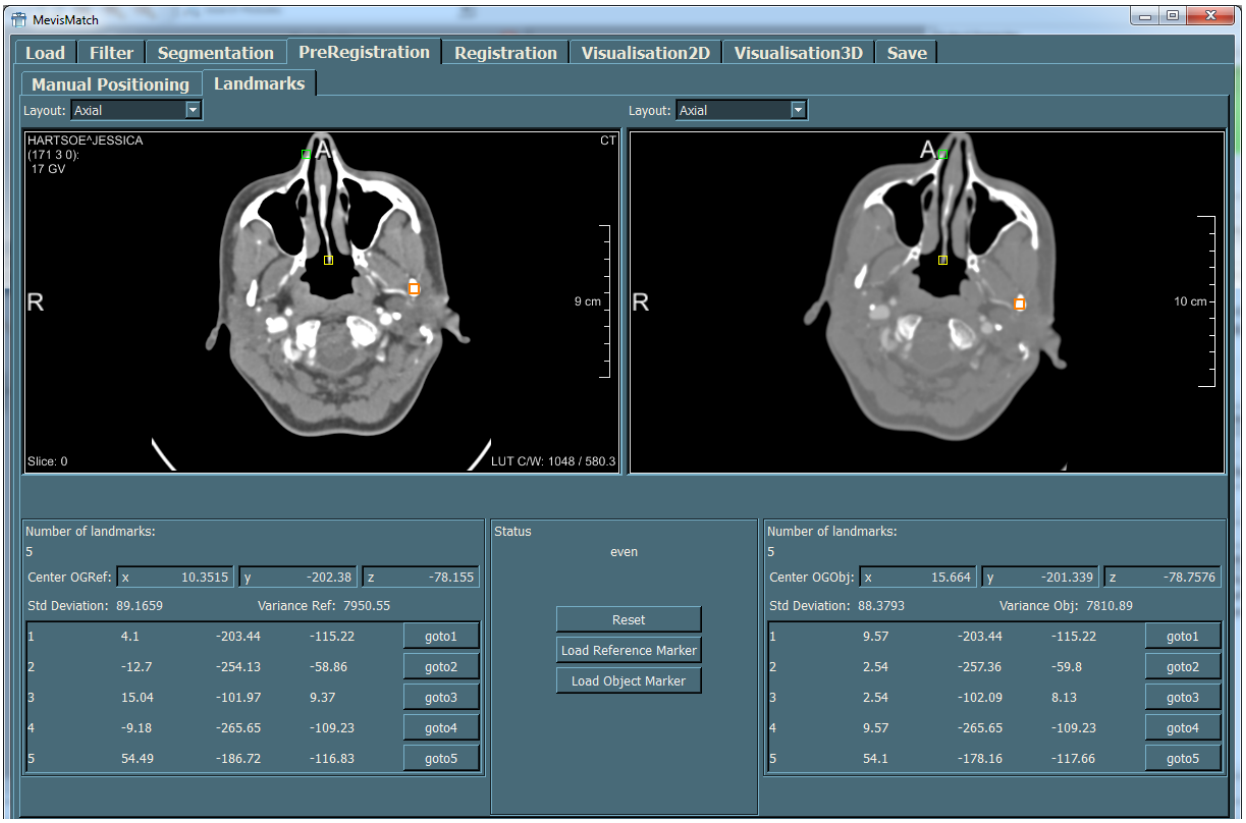


Abbildung 55: Tab zum Setzen der Landmarken. Hier können Landmarken paarweise gesetzt werden.

Unter **Landmarks** ist es möglich paarweise Landmarken zu setzen. Auf markante knöcherne Strukturen werden in diesem Beispiel nun jeweils 5 Landmarken gesetzt (s. Abbildung 55).

7.1.5 Registrierung

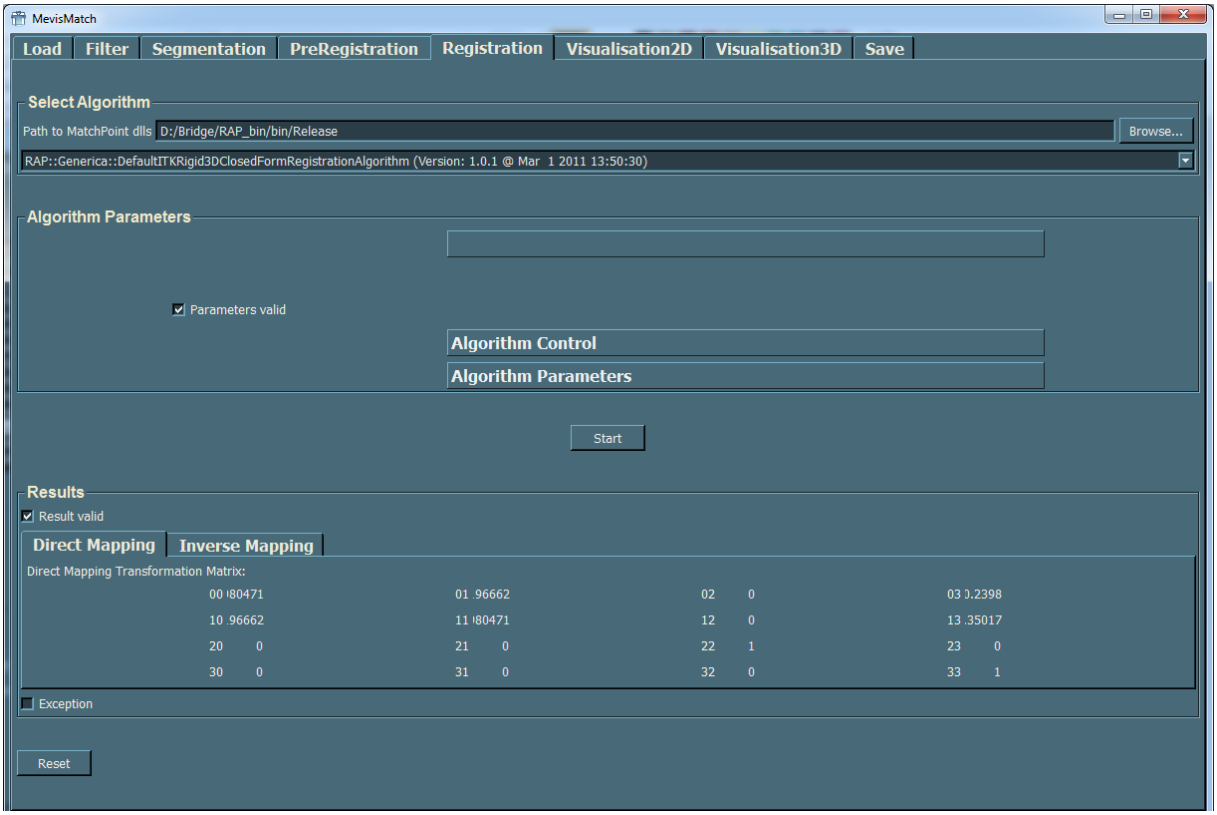


Abbildung 56: Tab für die Registrierung. Hierbei handelt es sich um die vereinfachte Benutzeroberfläche der Bridge. Sie wird unterteilt in die Algorithmenauswahl (oben), der Algorithmusparameter (mitte) und den Registrierungsergebnissen (unten)

Unter dem Menüpunkt **Registration** findet man die vereinfache Benutzeroberfläche der MatchPoint-MevisLab Bridge (s. Kapitel 2.2.5.1), dargestellt in Abbildung 56.

Im **Select Algorithm** Feld wird zuerst zum Ordner der Registrierungsbibliotheken navigiert und anschließend ein Registrierungsalgorithmus ausgewählt (s. Abbildung 57). In diesem Fall soll ein dreidimensionaler ClosedForm Algorithmus (s. Kapitel 2.1.5.1) angewandt werden. Fehlende oder fehlerhafte Parameter würden aufgezeigt und das Starten der Registrierung durch das Deaktivieren der entsprechenden Benutzeroberflächenelemente verhindert werden. Dies ist aber in diesem Beispiel nicht der Fall. Dementsprechend wird nun die Registrierung gestartet.

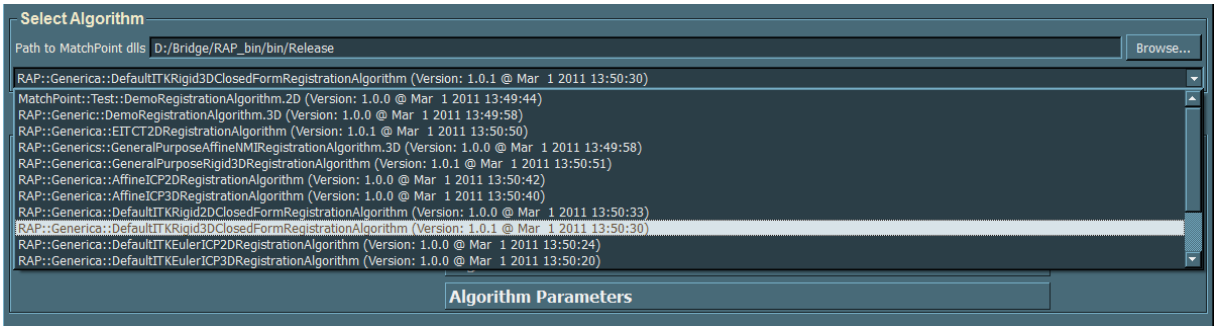


Abbildung 57: Auswahl von in MatchPoint entwickelten Registrierungsalgorithmen

Ist die Registrierung ohne Fehler verlaufen, wird im untersten Abschnitt die Transformationsmatrix dargestellt (s. Abbildung 56). Im Netzwerk wurden das Objektbild, die entsprechenden Landmarken und das segmentierte Bild bereits transformiert und in die entsprechenden Puffer geschrieben.

7.1.6 Visualisierung

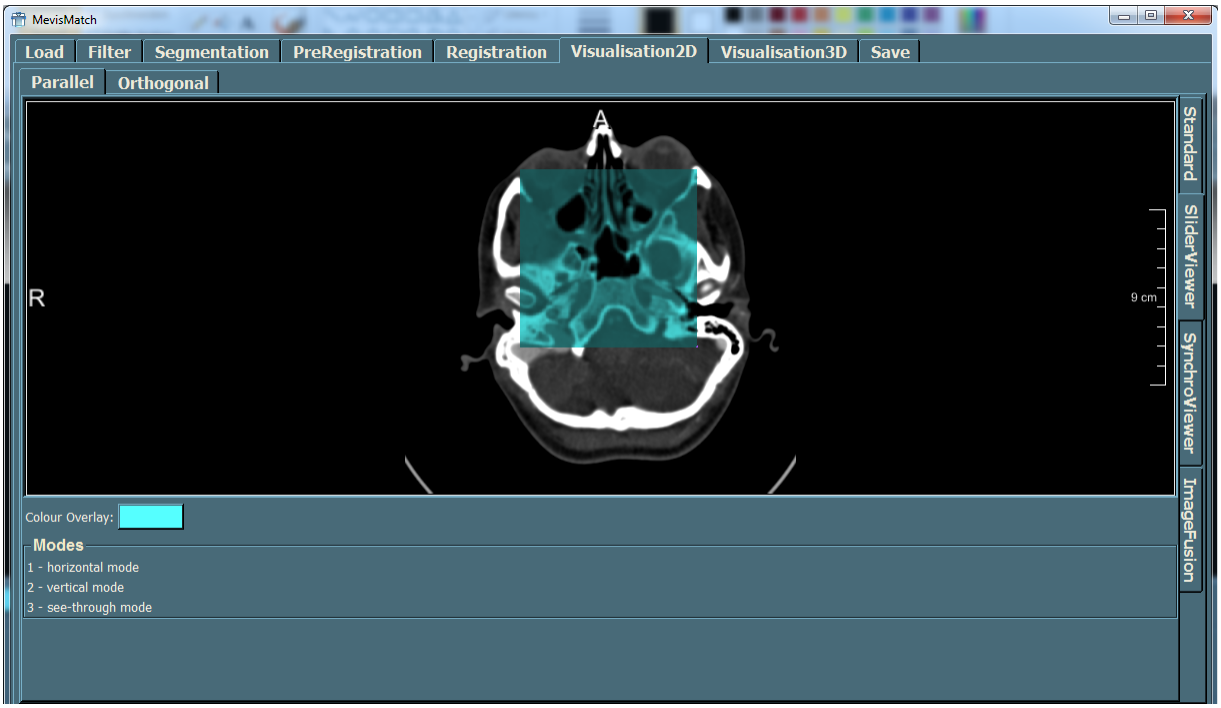


Abbildung 58: Das Slidermodul stellt sowohl das Referenzbild (weiss) als auch das Objektbild (grün) übereinander dar.

Unter Visualisation2D und 3D lassen sich die Bilddaten samt segmentierten Strukturen und Landmarken darstellen. Sie erlaubt die optische Evaluierung der Registrierungsergebnisse. Für die genaue Evaluierung der Bilddaten wird in Abbildung 58 die Visualisierung mit Hilfe des Slider-Moduls dargestellt (s. Kapitel 0). Dazu werden in Abbildung 59 die Datensätze vor und nach der Registrierung vergleichend dargestellt.

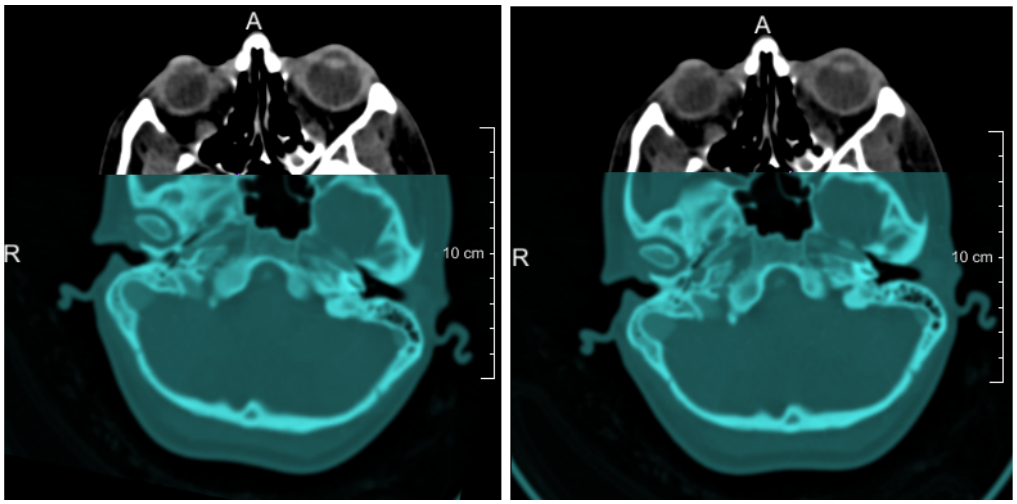


Abbildung 59: Vorher-Nachher-Ansicht der Bilddaten mit Hilfe des Slider-Moduls. Links - Ausgangsdatensätze, Rechts – Registrierte Datensätze.

7.1.7 Speichern

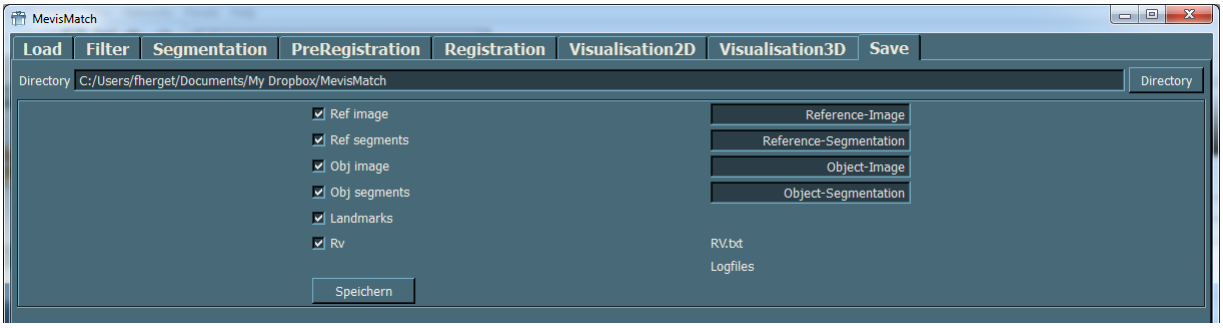


Abbildung 60: Tab zum Speichern der Bildatensätze und aller anderen gewonnenen Daten.

Im letzten Fenster lassen sich alle Bilddatensätze und Logfiles in einem beliebigen Ordner speichern. So finden sich anschließend alle gewonnenen Daten an einem zentralen Ort.

7.2 Alternative Arbeitsschritte

Die Netzwerkstruktur von MevisMatch wurde bewusst so aufgebaut, dass neben dem sequentiellen Bearbeiten der Bilder auch alternative Bearbeitungsvorgehen möglich sind. Verschiedene mögliche Bearbeitungssequenzen wären die folgenden:

Optimierung der Bilddaten zum Registrieren

Nach einer Registrierung wäre es möglich, die Ausgangsparameter wie Bilder, Landmarken oder segmentierte Strukturen zu modifizieren, um nach einer erneut durchgeführten Registrierung gegebenenfalls ein besseres Ergebnis zu erhalten.
Wie im vorherigen Durchlauf erwähnt, ist es auch möglich, durch Evaluierung der Lageverhältnisse zwischen den Strukturen bereits eine Vorauswahl bzgl. des zu verwendenden Algorithmus zu treffen.

Abstandsmessungen mit Hilfe der Landmarken

Aus den Positionen der gesetzten Landmarken werden bestimmte Kenngrößen berechnet, wie etwa der durchschnittliche Abstand zwischen den Landmarkenpaaren, die Varianz oder die Standardabweichung. Diese Werte können natürlich unabhängig von einer Registrierung zur Messung von Abständen zwischen zwei Punkten etc. verwendet werden.

Datenvorbereitung

Datensätze können für spätere Evaluierungen oder andere Anwendungen vorbereitet werden, so können z.B. Bilddaten gefiltert oder Strukturen segmentiert werden, um diese erst zu einem späteren Zeitpunkt oder mit einer anderen Anwendung weiterzuverarbeiten oder auszuwerten. Dazu wurde speziell darauf geachtet dass zu jedem Zeitpunkt die gewonnenen Daten in mehreren Formaten gespeichert werden können.

7.3 Erweiterbarkeit

Die einfache Erweiterbarkeit war ein Anspruch an die Applikation MevisMatch. Zukünftige Entwicklungen sollten einfach in die Applikation eingebunden werden können. Um eine Manipulation oder Erweiterung auf der Netzwerk-Ebene zu umgehen und die damit einhergehenden nötigen Änderungen an der Benutzeroberfläche, der Datenverwaltung und dem Scripting, wurde in Kapitel 5.2.1 beschlossen dass das Netzwerk aus Makromodulen aufgebaut werden muss. Daher wurden im Laufe der Entwicklung mehrere Makromodule für die Funktionalitäten angelegt, bei denen eine Erweiterung am Wahrscheinlichsten erschien (s. Kapitel 4.2.4). Dazu gehören die **Filter** (AllFilters), **Segmentierungsverfahren** (AllSegAlgorithms), **Registrierungsverfahren** (MPSWrapper) und die **Visualisierungsverfahren** (ViewAll2D und ViewAll3D). Die Prozedur des Erweiterns wurde dabei bewusst einfach und übersichtlich gehalten, sodass die Einarbeitungszeit in die Struktur der Module relativ kurz gehalten werden kann.

Im Folgenden werden die Erweiterungsverfahren der verschiedenen Modultypen erläutert.

Bei den Modulen **AllFilters** und **AllSegAlgorithms** wird ein neues Modul über drei Schritte integriert:

1. Hinzufügen des Moduls zum Netzwerk und Anbinden an den Switch (s. Abbildung 61)
2. Benutzeroberfläche in Script-Datei einbinden und Kopplung des ausgewählten Tabs an die Auswahl des Switches
3. Hinzufügen aller relevanten Parameter in das SettingsManager-Modul

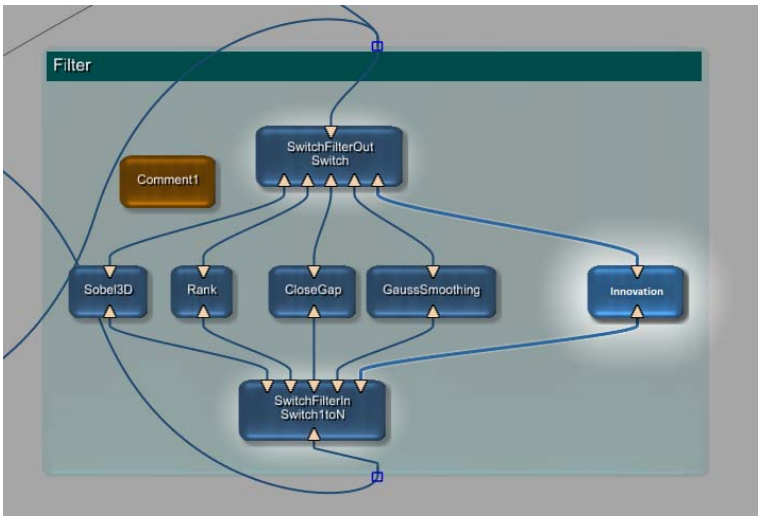


Abbildung 61: Das Modul Innovation wird dem Makromodul hinzugefügt

Zum Hinzufügen eines neuen **Viewers** zu den Modulen ViewAll2D oder ViewAll3D bedarf es ebenfalls nur drei Schritte:

1. Viewer an die Beipässe anschließen
2. Benutzeroberfläche in das Makromodul einbinden
3. Aktivierung bzw. Deaktivierung der Beipässe verwalten (s. Kapitel 5)

Die Erweiterung des Moduls **MPSWrapper** für die Registrierung kann auf zwei Ebenen erfolgen. Zum einen können außerhalb der Applikation neue MatchPoint Algorithmen erstellt und im Verlauf der Registrierung eingebunden werden. Zum anderen können innerhalb des Makromoduls MPSWrapper neue Module eingepflegt werden wie z.B. zusätzliche Registrierungsmodule.

Solange die Parameterwerte oder deren Namen sich nicht ändern, kann ein Update des Moduls **MPS** durchgeführt werden ohne dass die interne Netzstruktur von MPSWrapper geändert werden müsste.

7.4 Benutzeroberfläche

Zum Ende der Entwicklungsphase ergab eine Auswertung des Kapitels „Ergonomy and graphical user interface design“ von [Schulz98] , dass die entwickelte Applikation 19 von 23 Anforderungen an eine Software für die Medizin erfüllt. Diese Anforderungen behandeln beispielsweise die Übersichtlichkeit der Applikation, die Wiedererkennbarkeit von Funktionen oder die Konsistenz der Anwendung. Die ausgefüllte Liste samt Anmerkungen befindet sich im Anhang.

Zu den vier nicht erfüllten Punkten gehören zwei optionale Anforderungen, wie eine gedruckte Bedienungsanleitung und die Möglichkeit jederzeit aus dem Programm heraus drucken zu können. Einer der beiden weiteren nicht erfüllten Punkte betrifft die Hilfe-Funktionen. Innerhalb der Applikation wurde keine Hilfe-Funktionen implementiert, weil dies von MeVisLab nicht unterstützt wird. Stattdessen wurden mehrere externe Anleitungen geschrieben und für jedes Modul eine eigene Dokumentation angelegt (s. Kapitel 6.3).

Ein kritischer Punkt war, dass Farbe niemals der alleinige Träger von (Bild-)Informationen sein darf. Hintergrund dieser Anforderung ist die Rücksichtnahme auf Farbenblinde. Um diesen Aspekt erfüllen zu können, wurden bei einem Großteil der alten wie neuentwickelten Viewer die Möglichkeit gegeben, die Farben in denen die Strukturen dargestellt sind zu verändern. So können zumindest Anwender mit Rot-Grün-Schwäche oder allgemein partieller Farbenblindheit die Viewer nutzen.

7.5 Qualitätssicherung

Das im vorherigen Kapitel zu Rate gezogene Paper [Schulz98] beinhaltet auch ein Kapitel bzgl. der Softwareentwicklung. Nicht alle Aspekte können direkt auf MevisMatch bezogen werden, da es sich um keine eigenständige Anwendung handelt. Diese Punkte werden im Bezug auf MeVisLab beantwortet.
Die Anwendung erfüllt 11 von 15 vorgegebenen Punkten, wobei es sich bei zwei der nicht erfüllten Punkte um optionale Vorgaben handelt. Die komplette ausgewertete Tabelle befindet sich im Anhang. Die beiden noch offenen, nicht erfüllten Punkte werden im Folgenden erläutert.

Eine dieser Aspekte war die Frage, ob die Anwendung auf einem File-Server installiert werden kann, ohne dass Installationen auf der Seite des Clients erfolgen müssen. Da es sich bei MeVisLab um eine lokale Anwendung handelt die nicht darauf ausgelegt ist, über einem Server von mehreren Clients angesprochen zu werden, kann dieser Punkt nicht erfüllt werden.

Der letzte Aspekt betrifft die Anforderung “The application is **stable**, **robust** against improper use, **reliable** and of **good performance**”.
Die Anwendung an sich ist stabil, Berechnungen laufen ohne Fehler durch und es kommt zu keinen Ausfällen. Lediglich im Bezug auf die Viewer gibt es zeitweise Updateprobleme, besonders bei Zusammenstellungen mehrerer unabhängiger Viewer fallen einige aus und bleiben schwarz. Es scheint kein Muster zu geben nach denen die Viewer ausfallen, sie fallen auch unabhängig von der Auslastung des Rechners aus.
Ein anderes Problem betrifft die Anforderung „robust against improper use“. Wie in Kapitel 6.2 erläutert, muss MevisMatch aus MeVisLab heraus gestartet werden. Das heißt dass der Anwender potentiell auf das Netzwerk zugreifen und gegebenenfalls modifizieren kann.
Das Netzwerk wird verändert in dem Moment in dem die Anwendung gestartet wird. Das heißt dass der Anwender jedes Mal vor Beenden von MeVisLab abgefragt wird, ob er die Änderungen speichern will. Wurde das Netzwerk im größeren Ausmaß manipuliert, beispielsweise durch das Entfernen eines Moduls, kommt es nach dem Abspeichern zu Funktionseinbußen. Dementsprechend muss die Frage nach der Robustheit der Entwicklungsumgebung und somit der Anwendung mit Nein beantwortet werden.
Am Ende ist festzustellen, dass die beobachteten Mängel hauptsächlich auf die Entwicklungsumgebung zurückzuführen sind.

8 Diskussion und Ausblick

Das Ziel dieser Arbeit war die Entwicklung einer Applikation welche das Prä- und Postprocessing von Registrierungsverfahren unterstützt und als Entwicklungs- und Testumgebung für neue Algorithmen und Verfahren dient.
Nachfolgend werden die Ergebnisse dieser Arbeit hinsichtlich der erarbeiteten Konzepte, der Qualität und Funktionsvielfalt der Anwendung und einiger anderer Aspekte diskutiert.

Die gewählte **Netzwerkarchitektur**, basierend auf Puffer für die Bilddaten, erwies sich als richtiges Konzept für eine freie Abfolge von bildverarbeitenden Schritten. Die damit verbundene Freiheit führt aber auch zu einer höheren Komplexität. Dies betrifft nicht nur die Implementierung, bei der eine serielle Abfolge von bildverarbeitenden Schritten erheblich einfacher zu entwickeln gewesen wäre. Auch für den Anwender erweist sich das Einarbeiten in die Applikation als zeitaufwendiger.

Die **Entwicklung** der Netzwerkarchitektur konnte schon nach kurzer Zeit als abgeschlossen betrachtet werden, so gut wie alle Funktionen standen zur Verfügung oder wurden zeitnah entwickelt. Der weitaus größere Anteil der Entwicklungsphase betraf die Optimierung der zeitlichen Abläufe, das Entwickeln der Initialisierungsmethoden oder das Einrichten des Loggings.
Es musste festgestellt werden, dass die von MeVisLab bereitgestellten Dokumentationen nicht alle wichtigen Aspekte der Modul- und Netzwerkentwicklung abdecken. Die Dokumentationen an sich sind sehr ausführlich aber auch weitläufig und es fehlen bestimmte Zusammenhänge. Bestimmte Informationen sind schriftlich nicht festgehalten, bei Fragen konnte man sich allerdings immer an die Mitarbeiter von MeVis wenden.
Wie in Kapitel 6.2 erläutert fehlt zudem eine Möglichkeit, Netzwerke oder Module auf einen Ausgangszustand zurückzusetzen. Darum mussten für das Starten und das Beenden des Netzwerks sowie für fast jeden Funktionswechsel eigene Initialisierungsmethoden geschrieben und aufeinander abgestimmt werden. Dieser Aufwand hätte erheblich verringert werden können, gäbe es solch eine Funktion zum Zurücksetzen. Laut den Entwicklern von MeVisLab ist solch eine Funktion allerdings weder geplant noch wird sie als nötig erachtet, s. [MevF4] . Das Beibehalten von gewählten Parametern mag zwar auf der einen Seite ein Feature darstellen, kann aber auch zu fehlerhaften oder unerwarteten Ergebnissen führen.

Das **Erweiterungsprinzip** von Filter- und Segmentierungsverfahren hat sich im bisherigen Verlauf als gute Lösung herausgestellt. Neue Module sind durch zwei Klicks und zwei Zeilen Quellcode einfach in die Applikation einzubinden. Ebenso unkompliziert ist das Koppeln der Parameter an das Logging-Verfahren. Das gesamte Netzwerk ist aus Makromodulen aufgebaut, welche es ermöglichen alle inneren Bestandteile zu erweitern oder auszutauschen, ohne dass das Gesamtnetzwerk verändert werden muss.

Während der Entwicklung stellte sich heraus, dass je größer die Applikation und das dazugehörige Netzwerk, desto **instabiler** wurde die Anwendung. Da es auf Grund der gewählten Netzwerkarchitektur zu keiner Kaskade von neuen Berechnungen bei Feldänderungen kommen kann, war die Instabilität nicht auf eine zu hohe Belastung des Arbeitsspeichers zurückzuführen. Zudem waren hauptsächlich die Viewer von dieser Instabilität betroffen. Wie in Kapitel 6.2 erläutert stellte sich heraus dass das ungeordnete Feld und FieldListener System wohl der Grund hierfür ist. Die Viewer basieren auf Open Inventor Modulen, deren Berechnungen bei einer hohen

Rechnerauslastung zurückgestellt werden. Durch Feldänderungen bei ML-Modulen ausgelöste Neuberechnungen werden hingegen sofort und ohne Priorisierung durchgeführt. Diese Probleme konnten dahingehend eingegrenzt werden, als das die Anwendung MevisMatch hinsichtlich der Abfolge ihrer Prozesse und Kontrollstrukturen **optimiert** wurde. Hierzu wurden die verschiedenen Prozesse zeitlich aufeinander abgestimmt oder z.B. einzelne Viewer gezielt angesteuert, wie in Kapitel 0 erläutert. Es stellt sich die Frage, ob innerhalb des Frauenhofer Institutes neue Modulkonzepte oder andere Methoden zur Verfügung stehen. Die mit MeVisLab entwickelten Projekte für die Diagnostik und Therapeutik (s. [MevProj11]) laufen bewiesenermaßen stabil und entsprechen teilweise dem Medizinproduktegesetz [MPG94] .

Die obigen und andere in diesem Kontext auftretende Probleme sind eventuell auf die Qualitätssicherung zurückzuführen. Jeder Entwickler ist für die Stabilität seiner Module selber verantwortlich ([MeVisTut10] , Kapitel 12), außerdem scheint es, auch für die offiziellen Module, keine von außen erkennbaren **Qualitätsstandards** zu geben. Es stellt sich die Frage ob und wer die 1300 Module hinsichtlich Stabilität und Funktionsweise testet. Im Rahmen dieser Arbeit wurden gleich an mehreren Modulen gravierende Fehler erkannt, hierzu werden drei repräsentative Module vorgestellt.
Das Modul Thumbnail basiert auf veralteten Methoden und kämpft mit einem Updateproblem([MevF4]). Das Modul zum manuellen Registrieren RegistrationManual etwa scheint durch das in Kapitel 6 erläuterte Problem mit dem Abspeichern von Netzwerken aus Versehen manipuliert worden zu sein. Standardmäßig ist eine Rotation von 45° um die Z-Achse eingestellt(s. Abbildung 62). Dies entspricht zufälligerweise genau dem Grad Maß, um das im Beispielnetzwerk ein Bild registriert wird.

Abbildung 62: Auszug aus dem AutomaticPanel vom Modul RegistrationManual

Die von MeVisLab bereitgestellten Bestandteile der ITK-Registrierungsalgorithmen wurden automatisiert aus den ITK-Bibliotheken generiert [MeVF2] . Bei diesen Modulen fiel besonders auf, dass selbst bei der Verwendung identischer Datensätze die Berechnung eine solche Belastung des Arbeitsspeichers verursacht, ohne in vertretbarer Zeit Ergebnisse zu liefern, dass auf die Verwendung dieser Registrierungsmodule verzichtet werden musste.
Die Frage nach den Qualitätsmaßnahmen bei der Entwicklung von Modulen ist daher berechtigt.

Nachdem sich im Laufe der Arbeit immer mehr Defizite der Entwicklungsumgebung zu Tage kamen, stellte sich die Frage ob bei der Analyse die richtigen Faktoren und Quellen beachtet wurden. Im Rahmen der schriftlichen Ausarbeitung wurden daher noch einmal alle verwendeten Quellen für die evaluierten Entwicklungsumgebungen gesichtet. Lediglich in einem kleinen Absatz in [Heckel09] werden Stabilitätsprobleme erwähnt, aber auch nicht in diesem Kontext. Ebenso ist zu erwähnen dass in der Einleitung von MeVisLab eindeutig Multithreading aufgeführt wird, welches in einem aktuelleren Dokument negiert wird, siehe [MeVisTut10] Kapitel 2.1 „Prominent features […] Multithreading support“ versus [MevRef] Kapitel 23.7 „Multithreading is not really supported“.

Nach dieser Sichtung kam heraus, dass es keine schriftlichen Hinweise auf die beobachteten Mängel gab. Dabei muss aber erwähnt werden, dass bei Evaluierungen mehrerer Frameworks ([Bitter07] , [Rex05] , [Rex05a] etc.) nur kleine Applikationen entwickelt und getestet wurden. Stabilität und Performanz von MeVisLab wurde in keiner Quelle erörtert und das Thema kam auch bei persönlichen Gesprächen während der Analysephase nicht auf.
Auf Basis der zur Verfügung stehenden Daten und Informationen gab es keinen Grund sich für eine andere Entwicklungsumgebung zu entscheiden, allerdings hätte im Rahmen der Analysephase auf die Stabilität der entwickelten Anwendungen mehr eingegangen werden müssen.

Im Folgenden werden die in MevisMatch bereitgestellten bildverarbeitenden Verfahren diskutiert.

Für die **Registrierung** wurde zum Ende der Analysephase zwei Optionen in Aussicht gestellt, das Registrierungsaddon **MERIT** [MERIT] oder die MatchPoint-MevisLab-**Bridge**. Erstere konnte weder in der Einarbeitungsphase noch in der Implementierungsphase eingebunden werden, da MeVisLab erst kurz vor Abschluss der Implementierungsarbeiten MERIT zur Verfügung stellen konnte. Eine genaue Evaluierung war zu diesem Zeitpunkt nicht mehr möglich, zudem durfte das Addon nicht an dritte Personen weitergegeben werden. Daher wurde auf die Funktionen des Addons verzichtet. Dahingegen bestand gleich zu Beginn der Arbeit die Möglichkeit mit der Bridge zu arbeiten, samt einiger bereits implementierten Algorithmen von MatchPoint. Während der Entwicklungsphase wurde sie weiter ausgebaut und erlaubt es nun, alle affinen Registrierungsverfahren von MatchPoint anzuwenden. Die Umgebung zum Implementieren neuer Registrierungsverfahren wurde ebenfalls zur Verfügung gestellt.
Diese Kopplung an MatchPoint hat den Vorteil, dass im Anschluss an diese Arbeit das **Optimierungsframework** f.r.e.e. [Floca07] verwendet werden kann. Dabei handelt es sich um eine Bibliothek zur Optimierung von Registrierungsparametern.

Die **Evaluierung** der Registrierungsergebnisse erfolgt hauptsächlich über die visuelle Einschätzung des Anwenders. Alternativ dazu ist es möglich verschiedene Kennziffern der Landmarken auszuwerten und zu beurteilen, wie etwa der durchschnittliche Abstand der Landmarkenpaare, vor und nach der Registrierung. Weitere qualitative wie quantitative Kennziffern zur Bewertung der Registrierung wurden zunächst nicht implementiert, da diese abhängig vom gewählten Registrierungsverfahren berechnet werden müssen.
Hinsichtlich der **Visualisierung** und dementsprechend der **visuellen Evaluierung** konnten während dieser Arbeit verschiedene Konzepte realisiert werden, die noch nicht in MeVisLab entwickelt wurden. Dazu gehören die schon erwähnten Module Slider und ImageFusion. Diese Module wurden von Anwendern wie Entwicklern als nützlich eingestuft.

Leider war es, auf Grund der in Kapitel 4 erläuterten **Lizenzbeschränkungen** bzgl. der Modulanzahl, nicht möglich, den Umfang an bildverarbeitenden und evaluierenden Modulen beliebig groß zu gestalten. Viele Module aus dem Fundus der Arbeitsgruppe SIDT oder der MeVisLab-Community konnten nicht eingepflegt werden. Ebenso war es nicht möglich gewesen, aus MevisMatch eine **Standalone-Applikation** zu generieren. Wie in Kapitel 7 beschrieben sind die Nachteile für den Endanwender beträchtlich.

Obwohl zur Gestaltung der **Benutzeroberfläche** Programme aus dem Feld der Bildvorverarbeitung, Projekte in MeVisLab und [Schulz98] als Referenzen hinzugezogen wurden, hatten Tests mit zukünftigen Anwendern ergeben, dass diese mit der Orientierung innerhalb der Anwendung einige Probleme hatten. Dieser Umstand war zu erwarten und ein geeignetes Orientierungskonzept war bereits erarbeitet. Dies wird im Ausblick näher erläutert. Die zusammengestellten und neu entwickelten Funktionalitäten wurden hingegen als positiv und schlüssig konzipiert empfunden.

Neben der Orientierung war die bei den Filter- und Segmentierungsverfahren eingeführte Versionierung ein Problem für den Anwender. Wie in Kapitel 6.1.1 beschrieben werden in bis zu drei Viewern die verschiedenen Versionen des zu bearbeitenden Bildes dargestellt. Filter und Segmentierungsverfahren werden auf eine interne Kopie des Bilddatensatzes angewandt, erst wenn der Anwender mit dem Ergebnis zufrieden ist wird der alte Ausgangsdatensatz anhand eines speziellen Befehls überschrieben. Diese Unterscheidung zwischen temporären und dauerhaften Versionen des Bildes und den unterschiedlichen Formen des Speicherns führte zu einigen Verwechslungen.

Es ist bisher nicht möglich eine installierbare Version von MevisMatch zu erstellen, daher ist sie eher ungeeignet für eine Anwendung außerhalb des Instituts oder des DKFZ‘s. Die Installation ist langwierig und relativ komplex und würde einen Anwender überfordern. Zwar können für bestimmte Betriebssysteme die entsprechenden Bibliotheken vorbereitet werden, allerdings müssen diese im Nachhinein noch modifiziert werden. Für eine Neuinstallation existiert eine detaillierte Anleitung im Handbuch.

Ausblick

Die entwickelte Applikation befindet sich sicherlich noch im Anfangsstadium. Neben der schon erläuterten logischen Erweiterung um neue Filter-, Segmentierungs- und Registrierungsverfahren sind in Zukunft Erweiterungen auf struktureller Basis angebracht. Diese werden nun ihrer Wichtigkeit nach aufgeführt.

Laut der Projektgruppe um MatchPoint ist es geplant, sowohl MatchPoint als auch die Bridge um die **elastische Registrierung** zu erweitern. Um MevisMatch entsprechend anzupassen, muss lediglich das Netzwerk um ein Modul für die elastische Transformation erweitert werden. Dazu existieren in MeVisLab bereits zwei auf ITK basierende Module.

In [Heckel09] wird erwähnt das deren entwickelter Python-basierter Ansatz innerhalb von MeVisLab die **Stabilitätsprobleme** beheben kann, indem er das asynchrone FieldListener-System strukturiert und so große Anwendungen weniger anfällig für Fehler macht. Es wird in Aussicht gestellt, dass dieser Ansatz in einer der nächsten Versionen von MeVisLab eingepflegt werden soll. Da die Autoren dieser Publikation der Firma MeVis angehören, erscheint diese Quelle als verlässlich.

Hinsichtlich der **Benutzeroberfläche** sind Verbesserungsmöglichkeiten denkbar. Verschiedene (farbliche) Orientierungshilfen oder eine Wizardfunktion die den Anwender durch den Standardprozess der Bildverarbeitung leitet wären wünschenswert. Durch die Anbindung an Qt sind die Gestaltungsmöglichkeiten der Benutzeroberfläche relativ unbegrenzt.

Für die **Orientierung** innerhalb der Anwendung waren wie erwähnt schon einige Farbkonzepte erarbeitet. Geplant war entweder die unterschiedlichen Schritte farbig zu kodieren (s. Abbildung 63, hinten) und bzw. oder die Containerhierarchie in den einzelnen Tabs durch immer hellere Hintergründe voneinander abzusetzen (s. Abbildung 63, vorne).
Dies war allerdings nicht in dieser Form möglich, da MeVisLab nur Zugriffe auf die Farbe auf der Modulebene erlaubt. Da ein Großteil der Hauptcontainer aber auf der Netzwerkebene lag, konnte kein konsistentes Farbkonzept etabliert werden.
Dies hätte umgangen werden können wenn die komplette Applikation MevisMatch in ein Modul zusammengefasst worden wäre. Nun ist es aber so, dass die freie Version von MeVisLab (ohne Lizenz) es nur erlaubt Netzwerke mit höchstens 15 fremdentwickelten Modulen zu starten. Auf diese Zahl musste Rücksicht genommen werden, sollte doch die Applikation für eine zukünftige Anwendung im Institut ausgerichtet sein und Lizenzen auf Dauer nicht zur Verfügung stehen.

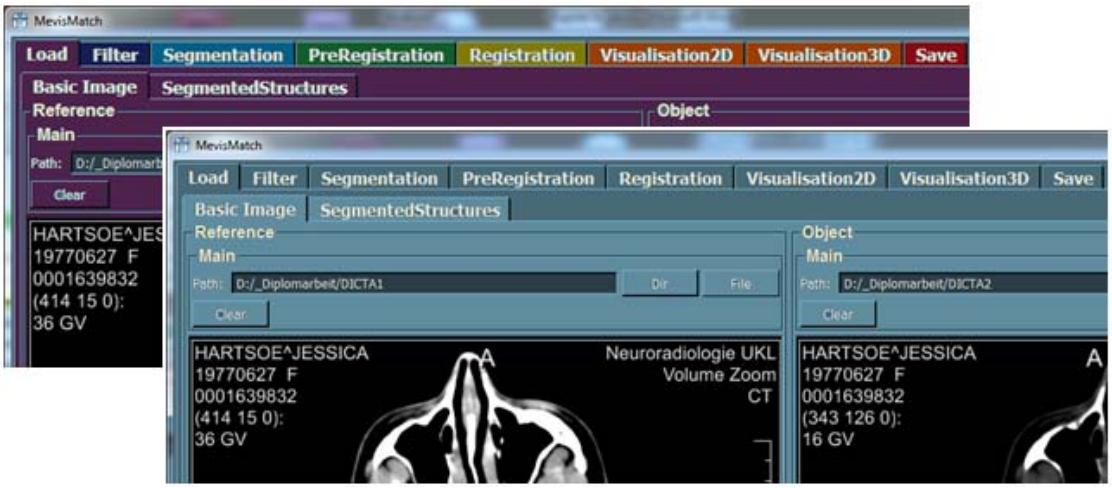


Abbildung 63: Farbkonzepte zur Orientierung innerhalb der Anwendung.
Hinten: Regenbogennavigation durch die verschiedenen Schritte
Vorne: Navigation durch die unterschiedlichen Ebenen in einem Tab

Die Implementierung der **Undo/Redo-Funktion** für die Filterung und Segmentierung wäre ein wichtiger Schritt hinsichtlich der Handhabung der Applikation. Dazu kann das verwendete Logging-Prinzip verwendet werden.

Es wäre wünschenswert wenn die Berechnungen für das Volume Rendering oder ähnlich aufwendige Berechnungen von MeVisLab auf die **Grafikkarte** ausgelagert werden würden, statt damit den Arbeitsspeicher zu belasten. Dies könnte mit Hilfe der Programmierplattform OpenCL (Open Computing Language, [Wiki4]) realisiert werden und würde das Laufzeitverhalten von MeVisLab erheblich verbessen.

Schon während dieser Arbeit kamen Anfragen für die entwickelte Applikation oder einige der Visualisierungsmodule, speziell aus der Arbeitsgruppe um MatchPoint. Es ist geplant MevisMatch dieser Arbeitsgruppe zur Verfügung zu stellen. Ebenso ist es angestrebt einzelne innovative Module, wie der Slider, der MeVisLab-Community anzubieten.

Insgesamt wurde während dieser Arbeit eine Anwendung geschaffen, die als Basis für zukünftige Arbeiten und Projekte im Bereich der medizinischen Bildverarbeitung und Registrierung fungieren kann.

Literaturverzeichnis

[Preim07] **Bernhard Preim, Dirk Bartz. 2007.** *Visualization in Medicine. Theory, Algorithms, and Applications : Theory, Algorithms, and Applications.* s.l. : Morgan Kaufmann, 2007.

[Birk11] **Birkfellner, Wolfgang. 2011.** *Applied Medical Image Processing.* New York : s.n., 2011.

[Bitter07] **Bitter, Ingmar, et al. 2006.** Comparison of Four Freely Available Frameworks for Image Processing and Visualization that use ITK. *IEEE TRANSACTIONS ON VISUALIZATION AND COMPUTER GRAPHICS.* 3, 2006, Bd. 13.

[Bühler07] **Bühler, Jonas. 2007.** Entwicklung eines Werkzeugs zur Koordinaten- und Grauwertinterpolation von MRI- und PET-Daten in der objektorientierten Umgebung MeVisLab. Fachhochschule Aachen, Labor für Medizinische Informatik : s.n., 2007.

[Floca09] **Floca, Ralf. 2009.** MatchPoint: On Bridging the Innovation Gap between Algorithmic Research and Clinical Use in Image Registration. *The World Congress on Medical Physics and Biomedical Engineering 2009.* München : s.n., 2009.

[Floca07] **Floca, Ralf und Dickhaus, Hartmut. 2007.** A Flexible Registration and Evaluation Engine (f.r.e.e.). *Computer Methods and Programs in Biomedicine.* 2007.

[Heckel09] **Frank Heckel, Michael Schwier and Heinz-Otto Peitgen. 2009.** *Object-oriented application development with MeVisLab and Python.* 2009.

[MPG94] Gesetz über Medizinprodukte - MPG. Berlin : s.n., 1994.

[Handels09] **Handels, Heinz. 2009.** *Medizinische Bildverarbeitung: Bildanalyse, Mustererkennung und Visualisierung für die computergestützte ärztliche Diagnostik und Therapie.* s.l. : Vieweg+Teubner, 2009.

[Heise1] Heiseartikel zu: Kitware offers free access to VolView. [Online] [Zitat vom: 14. 02 2011.] http://www.kitware.com/news/home/browse/Paraview?2010_11_29&Kitware+Offers+Free+Access+to+VolView.

[Heise2] Heiseartikel zu: SGI gibt Open Inventor 3D als Open Source frei. [Online] [Zitat vom: 22. 02 2011.] http://www.heise.de/newsticker/meldung/SGI-gibt-Open-Inventor-3D-als-Open-Source-frei-28041.html.

[IRTK] IRTK - Image Registration Toolkit. [Online] [Zitat vom: 14. 08 2010.] http://www.doc.ic.ac.uk/~dr/software/.

[ITK] ITK - Insight Segmentation & Registration Toolkit. [Online] [Zitat vom: 14. 08 2010.] www.itk.org.

[Jähne10] **Jähne, Bernd. 2010.** *Digitale Bildverarbeitung.* Berlin : Springer, 2010. Bd. Kapitel 16.

[Suri05] **Jasjit S. Suri, David L. Wilson und Swamy Laxminarayan. 2005.** *Handbook of Biomedical Image Analysis Volume III: Registration Models.* Berlin : Springer, 2005. Bd. 3.

[Lehmann97] **Lehmann, Thomas und Oberschelp, Walter. 1997.** *Bildverarbeitung für die Medizin.* Berlin : Springer Verlag, 1997.

[MevF3] **Link, Florian. 2010.** MeVisLab Forumsbeitrag zu ADK-Lizenzen. [Online] 07. 01 2010. [Zitat vom: 04. 09 2010.] http://www.mevis-research.de/cgi-bin/discus/board-auth.cgi?file=/839/10424.html.

[Ibanez03] **Luis Ibanez, Will Schroeder, Lydia Nq, Josh Cates. 2003.** *The ITK Software Guide: The Insight Segmentation and Registration Toolkit.* s.l. : Kitware Inc., 2003.

[MERIT] MERIT. [Online] [Zitat vom: 14. 10 2010.] http://www.mevis.de/mre/merit.html.

[MeVisLab] MeVisLab. [Online] [Zitat vom: 12. 08 2011.] www.mevislab.de.

[MevRef] MeVisLab - Reference Guide for the MeVisLab GUI. [Online] 06 2010. [Zitat vom: 16. 02 2011.]

[MevF4] MeVisLab Forum - Beitrag bzgl. des Resetten des Netzwerks. [Online] 15. 10 2010. [Zitat vom: 2011. 05 09.] http://www.mevis-research.de/cgi-bin/discus/board-auth.cgi?file=/839/12402.html&lm=1288168747.

[MeVF2] MeVisLab Forum - Beitrag zu ITK Modulen. [Online] http://www.mevis-research.de/cgi-bin/discus/board-auth.cgi?file=/1980/2168.html.

[MeVF1] MeVisLab Forum - Beitrag zur Weiterberechnung im Bildverarbeitenden netzwerk. [Online] http://www.mevis-research.de/cgi-bin/discus/board-auth.cgi?file=/841/8143.html&lm=1236272187.

[MevLic11] MeVisLab License Requirements. [Online] [Zitat vom: 12. 04 2011.] http://www.mevislab.de/mevislab/versions-and-licensing/.

[MevProj11] MeVisLab Projekte des Fraunhofer Instituts. [Online] [Zitat vom: 09. 05 2011.] http://www.mevis.fraunhofer.de/Projekte.html.

[MITK] **MITK**. [Online] [Zitat vom: 12. 09 2010.] http://mitk.org/.

[MITK1] **MITK 3M3 - Bilddatenviewer**. [Online] http://mint-medical.de/productssolutions/mitk3m3/mitk3m3/.

[Neff05] **Neff, Thomas. 2005.** Verbesserung der Zielvolumendefinition in der Strahlentherapieplanung durch den Einsatz der biologischen Bildgebung. Mannheim : s.n., 2005.

[Ope10] **Open Inventor**. [Online] [Zitat vom: 15. 09 2010.] http://oss.sgi.com/projects/inventor/.

[Prüm11] **Prüm, Hermann. 2011.** Aufbau eines Softwaresystems zur multimodalen Registrierung medizinischer Bilddaten und Verifikation von elastischen multimodalen Registrierungsverfahren. Heidelberg : s.n., 2011.

[Rex05] **Rexilius J, Spindler W, Jomier J, Link F, König K, and Peitgen HO. 2005.** Efficient Algorithm Evaluation and Rapid Prototyping of Clinical Applications using ITK. Chicago : s.n., 2005.

[Rex05a] **Rexilius, Jan, et al. 2005.** Combining a Visual Programming and Rapid Prototyping Platform with ITK. *Proc. Workshop Bildverarbeitung für die Medizin.* Heidelberg : Springer Verlag, 2005.

[Schulz98] **S. SCHULZ, T. AUHUBER, U. SCHRADER, R. KLAR. 1998.** Quality Criteria for Electronic Publications in Medicine. s.l. : University Hospital Freiburg, Department of Medical Informatics, 1998.

[Schlegl07] **Schlegl, Wolfgang. 2007.** *3D Conformal Radiation Therapy: Multimedia Introduction to Methods and Techniques.* Berlin : Springer, 2007.

[Tschum10] **Tschumperlé, David. 2010.** CIMG - Template Image Processing Toolkit. 2010.

[MeVisTut10] Tutorial zum Arbeiten mit MeVisLab - Getting Started. 2010.

[Vol10] **VolView**. [Online] [Zitat vom: 12. 08 2010.] www.volview.com.

[Wiki1] Wikipedia Artikel zu MITK. [Online] [Zitat vom: 02. 05 2011.] http://de.wikipedia.org/wiki/MITK.

[Wiki2] Wikipedia Artikel zu Open Inventor. [Online] [Zitat vom: 22. 02 2011.] http://de.wikipedia.org/wiki/Open_Inventor.

[Wiki4] Wikipedia Artikel zu OpenCL. [Online] [Zitat vom: 09. 05 2011.] http://de.wikipedia.org/wiki/Open_CL.

[Wiki3] Wikipedia Artikel zur Ebene im Raum. [Online] http://de.wikipedia.org/wiki/Ebene_%28Mathematik%29.

[Wels96] **William M. Wells III, Paul Viola, Hideki Atsumi, Shin Nakajima, Ron Kikinis. 1996.** Multi-Modal Volume Registration by Maximization of Mutual Information. [Buchverf.] Elsevier B.V. *Medical Image Analysis.* 1996.

Glossar

Wort	Definition	Kapitel
Bypass (Bypass)	Modul welches als Schalter fungiert und Daten an mehrere andere Module verteilt	4
CT	Computertomographie	-
CTA	Computertomographische Angiographie	-
DICOM	Standard zum Speichern und Versenden von Bilddatensätzen in der Medizin	2
Grauwert, Intensitätswert	Der Grauwert beschreibt wie hell ein Bildpunkt für das menschliche Auge erscheint	-
Landmarken	Vom Anwender markierte, anatomisch eindeutige Punkte	2
Makromodul	Modultyp in MeVisLab, der es ermöglicht mehrere andere Module zusammenzufassen	4
MPR	Multi Planar Resolution – Schnitt durch einen dreidimensionalen Bilddatensatz in beliebigem Winkel	2
Objektbild	Bild welches bei der Registrierung transformiert wird	2
Orthoviewer, Orthogonalviewer	Viewer	2
Overlay	Überlagerung eines Bilde mit anderen(Bild-) Informationen	6
Referenzbild	Das feste Bild, auf dessen Koordinatensystem bei der Registrierung transformiert wird	2
Serie	Alle Schichtaufnahmen einer dreidimensionalen Aufnahme	2
Standardsichten	Die drei Ansichten frontal, saggital und transversal	2
Viewer	Modul zum Sichten eines Bilddatensatzes	2

Abbildungsverzeichnis

Anhang

Auswertungen anhand des Papers [Schulz98]

Die Anforderungen an die entwickelte Software und Benutzeroberfläche wurden anhand der **Quality Criteria for Electronic Publications in Medicine** von [Schulz98] zusammengestellt und nach Abschluss der Entwicklungsphase bewertet.

Tabelle 6: Bewertung der erfüllten Punkte

✓	Ja
×	Nein
○	Nicht nötig oder vorhanden

Anforderungen an die Software

ML	MM	Anforderungen
✓	✓	The system requirements are clearly identified.
✓	×	The application is stable, robust against improper use, reliable and of good performance.
○	○	Basic functions of the application can (optionally) be started directly from the data medium without setup routine.
✓	✓	The application runs without modifying system areas of the operating system.
✓	✓	There is no need of system reboot or manual modification of the configuration to start the application the first time.
✓	○	Where an installation routine cannot be avoided, all system modifications are clearly documented and an uninstall routine is available.
✓	✓	The application is developed for the most common platform among target users. Ideally more than one platform is supported and the same functionality is available on any platform.
✓	✓	Special display properties (fixed display resolution or colour settings) are only acceptable if the graphical contents of the software makes it unavoidable.
×	×	The software can be installed on a file server without the need of separate client installations.
○	○	The software (this applies mainly for databases) runs as a real client / server application with clients for different platforms.
✓	○	Installation on a multi-user operating system is supported.
○	○	Multi-user operation supports storing of user-specific settings.
✓	✓	The application does not affront the users' patience by long loading times.
○	✓	Where there re considerable response delays, a warning is displayed.
✓	✓	No special computer skills are necessary.
✓	○	Interfaces for linking with complex systems (hospital information systems, text retrieval systems) are defined and documentation is available.
11	**8**	**von 16 Punkten**

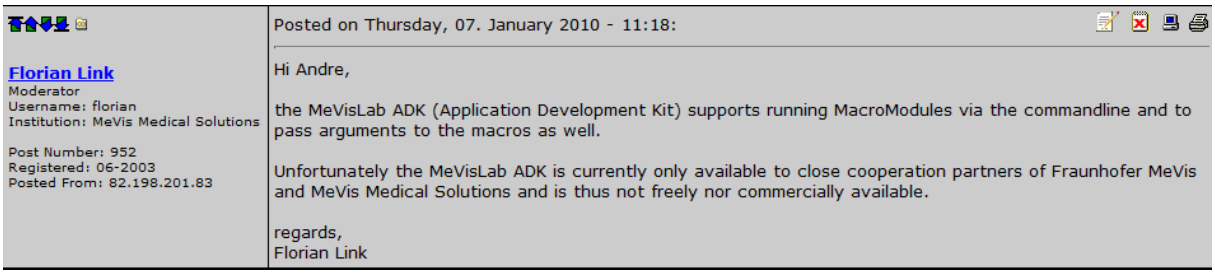
Anforderungen an die Benutzeroberfläche

	Anforderungen
✓	The GUI is guided by standards familiar to the user.
✓	Especially the following features should be modelled after wellknown standard applications (Office software, WWW browsers, Mail programs, functions of the operating system): - navigation tools in hypertexts - controls for playing audio and video sequences (tape-deck metaphor) - saving of user-specific options - file operations - retrieval functions - functionality of mouse buttons.
✓	Number and variety of controls (buttons etc.) is limited to the necessary.
✓	Controls on the main window always have the same position and continue to stay visible when disabled. They are clearly identifiable.
✓	The functions of the control elements are explained in an on-line help module.
✓	It is always evident which areas on the screen are sensible to mouse click and which are not.
✓	Mouse functions can alternatively invoked by key shortcuts.
✓	The application can be ended at any time and at any place.
✓	Functions that are expected to be used less frequently, are invoked by additional windows or through the menu bar, but not directly by buttons on the main window. Standardised windows contain always the same kind of information.
✓	The number of simultaneously open windows is limited.
✓	The more user interaction needed, the more necessary is the use of common GUI standards.
✓	If the GUI is has an uncommon and experimental design, the same creative philosophy applies consequently to the whole of the application.
✓	The colours of backgrounds, forms and controls are discrete and unobtrusive. Colours do not complicate the legibility of text and the usability of graphical elements.
×	Colour is never used as the only carrier of information (colour-blindness!)[2]
✓	The symbolism of colours and fonts is consistent and modelled after standard applications.
✓	Icons use metaphors that are plausible for the target users. Icons familiar from standard software are used if possible.
✓	The function of control buttons is always evident.
○	Texts and graphics can be printed at any time.
✓	The choice of stylistic and decorative elements depends on the target users and motivates them to use the application.
✓	The basic functions of the application can be handled without consultation of manuals or the help function.
○	The availability of a printed manual is not necessarily a quality criterion. If it exists it is complete, correct and user-friendly.
✓	All manual entries can be alternatively invoked out of the application and can be printed.
×	Help functions are context sensitive, printable and searchable. The hypertext criteria (above) apply also to help functions.
19/ 23	**Ergebnis**

[2] Eingeschränkt erfüllt, siehe Kapitel 7.4

Screenshots

Lizensierung MeVisLab



Florian Link
Moderator
Username: florian
Institution: MeVis Medical Solutions

Post Number: 952
Registered: 06-2003
Posted From: 82.198.201.83

Posted on Thursday, 07. January 2010 - 11:18:

Hi Andre,

the MeVisLab ADK (Application Development Kit) supports running MacroModules via the commandline and to pass arguments to the macros as well.

Unfortunately the MeVisLab ADK is currently only available to close cooperation partners of Fraunhofer MeVis and MeVis Medical Solutions and is thus not freely nor commercially available.

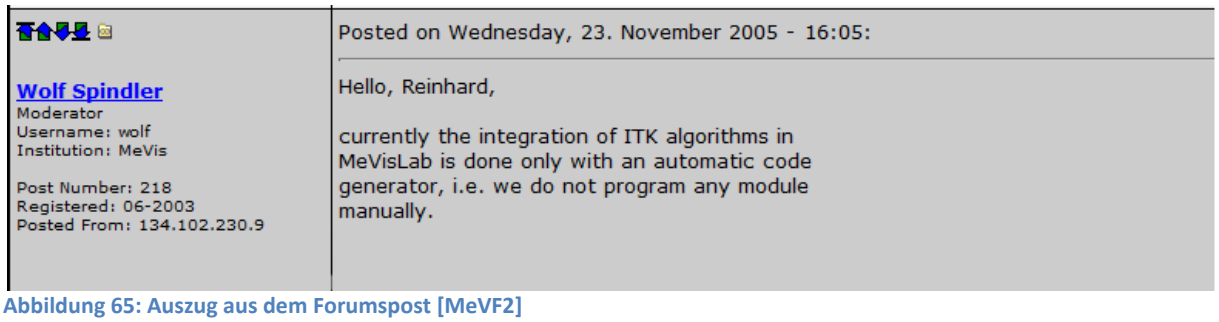
regards,
Florian Link

Abbildung 64: Auszug aus dem Forumspost [MevF3]

Feature	Commercial MeVisLab SDK	Non-commercial MeVisLab SDK	MeVisLab SDK Evaluation	MeVisLab SDK Unregistered
Unrestricted access to the complete MeVisLab SDK module set	+	+	+	+
Develop C++ modules (ML, Open Inventor, ITK)	+	+	+	+
Develop macro modules	+	+	+	+
Sign own modules	+	+	-	-
Use signed third-party modules	+	+	+	+
Use own or unsigned third-party modules	+	+	+	Instantiate up to 15 modules at a time
			Panels show license disclaimer	
Use scripting in own or unsigned third-party modules	+	+	+	Inline scripting only
Encrypt own modules	+	-	-	-
Use in commercial organizations	+	-	+	-
Use in non-commercial organizations	+	+	+	+
Distribute own modules	+	For research and evaluation purposes, not for commercial products or services		
Redistribute MeVisLab runtime environment	-	-	-	-
Available for free	-	-	+	+

Tabelle 7: Lizenzen [MeVisLab]

ITK-Module in MeVisLab



Wolf Spindler
Moderator
Username: wolf
Institution: MeVis

Post Number: 218
Registered: 06-2003
Posted From: 134.102.230.9

Posted on Wednesday, 23. November 2005 - 16:05:

Hello, Reinhard,

currently the integration of ITK algorithms in MeVisLab is done only with an automatic code generator, i.e. we do not program any module manually.

Abbildung 65: Auszug aus dem Forumspost [MeVF2]

Multithreading

2.1. MeVisLab Basics

Some of the most prominent features of MeVisLab:

- Full 6D image processing (x, y, z, color, time, user dimensions)
- Paging
- Caching
- Multithreading support
- Platform-independent
- Scripting support (Python and JavaScript)
- Macro system
- Defining of GUI elements with the MDL scripting language

Abbildung 66: [MeVisTut10] S.15

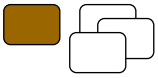
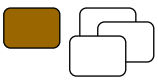
23.7.1. ML Multithreading

Currently, ML multithreading is not really supported. The ML itself limits multithreading at several points:

- It limits the maximum amount of pages processed in parallel, to avoid that too much memory is used at once,

Abbildung 67: [MevRef] Kapitel 23.7

Dateien der verschiedenen Modultypen

Dateityp	Inhalte	Module
`.def`	Modul Definitionsdatei, notwendig um das Modul zur MeVisLab Moduldatenbank hinzuzufügen. Beinhaltet manchmal die Definition der Benutzeroberfläche.	
`.script`	Datei für die Definition der Benutzeroberfläche mit Hilfe der internen Skriptsprache MDL (Module Definition Language)	
`.vcproj`	Projektdatei für die Implementierung von Modulen mit C++.	
`.mlab`	Netzwerk Datei, beinhaltet alle Module samt deren Parametern und ihren Verbindungen zu anderen Modulen.	
`.py`	Python Datei, wird für das Implementieren von Methoden und zur dynamischen Verwaltung des Netzwerkes benötigt.	
`.js`	JavaScript Datei, wird für das Implementieren von Methoden und zur dynamischen Verwaltung des Netzwerkes benötigt.