HTL - Perg
Höhere Abteilung für Informatik

Diplomarbeit

Medical Learning Assistant

Projektteam: Alexander Kranl
Andreas Höbarth
Projektbetreuer: Prof. Dipl.-Ing. Dr. Michael Buchberger

In Zusammenarbeit mit der FH Gesundheitsberufe OÖ

Bearbeitungszeitraum: 01.10.2014 – 13.05.201 5

HTL Perg | Machlandstr. 48 | A-4320 Perg | Tel.: +43 (0) 7262 53926

Eidesstattliche Erklärung

Hiermit versichern wir, die vorliegende Arbeit selbständig, ohne fremde Hilfe und ohne Benutzung anderer als der von uns angegebenen Quellen angefertigt zu haben. Alle Stellen, die wörtlich oder sinngemäß aus fremden Quellen direkt oder indirekt übernommen wurden, sind als solche gekennzeichnet.

Perg, _____ Unterschrift _____

(Alexander Kranl)

Perg, _____ Unterschrift _____

(Andreas Höbarth)

Kurzfassung

Die Diplomarbeit, „Medical Learning Assistant (MLA)“, dient als Lernhilfe für Studenten in medizinischen Ausbildungseinrichtungen. Dabei wurde eine Anwendung implementiert, die sich in ein bereits bestehendes Programm in Form eines Moduls einbinden lässt. Diese Lernhilfe informiert über medizinische Bildverarbeitungsverfahren und deren Anwendung in der Praxis. Die Anleitungen, die aus der Zusammenarbeit mit der Fachhochschule für Gesundheitsberufe in Form von Word Dokumenten entstanden sind, werden im Programm eingelesen, um sie daraufhin in eine visuell ansprechende Oberfläche zu verwandeln. Unter anderem befinden sich in den Anleitungen Vorgänge wie die Segmentierung einer Nierenschiene oder eines Gehirns sowie die Anwendung von verschiedensten Filtern zur Darstellung. Damit diese Anleitungen verfasst werden konnten, war es notwendig, Beispieldaten von CT, MRT und Röntgengeräten zu erfassen. Im Zuge dessen wurde ein Schädelscan im MRT anhand einer Testperson durchgeführt, um Testdaten zu akquirieren.

Abstract

The thesis “Medical Learning Assistant” (MLA) serves as a learning aid for students in medical education institutions. In this case, an application has been implemented, which can be integrated into an existing program like a module. This learning aid provides information on medical image processing and its application in practice. The instructions that have emerged from the collaboration with the College of Health Professions in the form of Word documents are parsed in the program, then transformed into a visually appealing interface. Among others, the instructions contain processes such as the segmentation of a kidney rail or a brain and the use of various filters for display. In order to write those instructions, it was necessary to capture sample data of CT, MRI and X-ray machines. As a result, a MRI scan of the skull was performed using a test person to acquire test data.

Danksagung

Wir bedanken uns bei all jenen, die uns im Laufe der Diplomarbeit unterstützt haben, sehr herzlich.
Besonderen Dank möchten wir gegenüber Dipl.-Ing. Dr. Michael Buchberger kundtun, da er die Diplomarbeit ins Leben rief. Unser Dank gilt ebenfalls unseren Kooperationspartnern, Patrick Knogler und Johannes Panzenböck, Studenten der FH Gesundheitsberufe, die uns mit zahlreichen Fallbeispielen versorgt haben.

Inhaltsverzeichnis

Abbildungsverzeichnis

Codebeispielverzeichnis

1. Einleitung

1.1. Problemstellung

Der Themenkomplex der medizinischen Bildgebung und Verarbeitung ist subjektiv betrachtet sehr umfangreich und kompliziert. Speziell junge Studenten wissen oft nicht wo und wann man die jeweiligen Verfahren und Methoden verwendet, um das gewünschte Ergebnis zu erhalten. Darum ist der Einstieg in das Thema oft mühevoll und bringt einige zur Verzweiflung.

1.2. Zielsetzung

Der Unterricht soll unterstützt werden, ebenso soll das Thema leicht verständlich dargestellt werden. Dabei soll für die Studenten des Studiums Radiologietechnologie mit dem Themenbereich medizinische Bildverarbeitung das Selbststudium unterstützt werden. Der Unterricht soll interaktiver gestaltet werden, indem die Inhalte teilweise mit Unterstützung des Programmes wiederholt bzw. präsentiert werden.

2. Medizinische Grundlagen

2.1. Tomographie

Die Tomographie (von altgriechisch τομή, tome, „Schnitt“ und γράφειν, grafëin, „schreiben“) beschreibt bildgebende Verfahren, welche Objekte in dreidimensionale Bilddatensätze überführt. Dies erfolgt in Form von **Schnittbildern** (auch Schichtbilder oder Tomogramme genannt) (siehe **Kapitel 2.2**).

Mithilfe der tomographischen Techniken können die innere Struktur von Materialien und Organismen, der Aufbau komplexer Bauteile der Mikrotechnologie oder die allgemeine, geometrische Struktur von Objekten dargestellt werden [Technische Universität Dresden].

2.1.1. Anwendung der Tomographie in der Medizin

- **Röntgentomografie (veraltet)**
  Diese basiert auf der konventionellen Röntgenuntersuchung, jedoch werden nur die Anteile im Körper scharf dargestellt, die untersucht werden sollen, der Rest wird verwischt [www.uni-bonn-radiologie.de].
- **Sonografie**
  Mithilfe von Ultraschallwellen, die von einem Schallkopf ausgesandt und wieder empfangen werden, kann durch die Reflektion der Schallwellen ein Graustufenbild erstellt werden [www.uni-bonn-radiologie.de].
- **Positronen-Emissions-Tomografie**
  Wird oftmals gemeinsam mit CT verwendet, da es ebenfalls eine radioaktive Substanz, die in den Körper gespritzt wird, sichtbar macht [Wikipedia].
- **Computertomografie**
  Ein anderes, ähnliches Konzept wäre z.B. die Magnetresonanztomographie

2.2. Schnittbilder

Ein Schnittbild gibt die inneren Strukturen so wieder, wie sie nach dem Aufschneiden eines Objektes oder nach dem Herausschneiden einer dünnen Scheibe vorläge. Im Gegensatz zum Projektionsverfahren (alle Strukturen eines Objekts überlagern sich) kann bei Schnittbildern eine überlagerungsfreie Darstellung garantiert werden.

Das untersuchte Objekt wird beim Schnittbildverfahren schichtweise gescannt. Die dabei entstehenden 2D-Schnittbilder bestehen aus einem Raster von Bildelementen, den Pixeln, wie sie auch in konventionellen Bildern vorkommen.

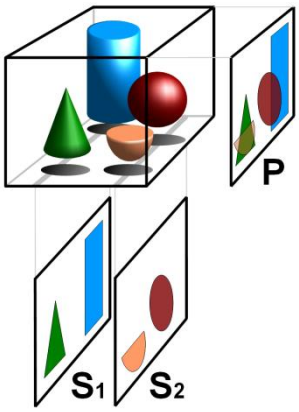

Abb. 2.1 Projektionsverfahren und Schnittbildverfahren

In der **Abb. 2.1** ist dargestellt, wie das Objekt mithilfe des Projektionsverfahren (P) aussehen würde. S1 und S2 beschreiben, wie das Objekt mithilfe des Schnittbildverfahrens aussieht.

Mithilfe eines Computers können diese einzelnen Schichten nun zu einem 3D Modell zusammengefügt werden [ANTWERPES, Frank et al.] [radiologie.de, 2013].

2.3. CT

2.3.1. Allgemein

Die Computertomographie (CT) ist ein auf Röntgenstrahlung basierendes Verfahren zur Erzeugung von Schnittbildern eines Objektes, die am Computer errechnet und zu einem dreidimensionalen Volumendatensatz zusammengefasst werden können. Haupteinsatzgebiet der CT ist die Medizin, aber auch in anderen Bereichen wie etwa der Archäologie, der Paläontologie, der Forensik oder in Prüfungen von Materialien wird diese Technik heute eingesetzt.

Im Gegensatz zur Röntgentomographie ist bei der CT die Nutzung eines Computers zwingend notwendig, daher auch der Name.

Die ersten mathematischen Grundlagen dazu brachte 1917 Johann Radon mittels seiner Radontransformation, welche zur Berechnung räumlicher Aufnahmen eines Objekts aufgrund seiner Durchdringbarkeit durch Röntgenstrahlung dient.

Erst im Jahre 1969 entstand der erste Prototyp eines CT durch Godfrey Hounsfield, Angestellter der Firma EMI. 1972 kam es schlussendlich zur Markreife, da bis dahin die Geräte als unbedenklich eingestuft wurden. Nun war es zum ersten Mal möglich, überlagerungsfreie Schichtbilder vom menschlichen Körper zu machen.

Die Computertomographie ermöglicht es vor allem, eher härteres Gewebe (da dieses mehr Strahlen abfängt - z.B. Knochen) darzustellen, weiches Gewebe

hingegen wird weniger gut aufgelöst, jedoch kann mittels Kontrastmitteln Abhilfe geschaffen werden.

2.3.2. Funktionsweise

Die wesentlichen Bestandteile eines Computertomographen sind, wie in **Abb. 2.2** ersichtlich, der Patientenlagerungstisch, ein Bedienpult mit Auswerteeinheit, Rechner, Bildwiedergabeeinrichtung, Archivspeicher und Gantry (=Fassöffnung). Die Fassöffnung enthält die Röntgenröhre, Detektoren und das Lichtvisier.

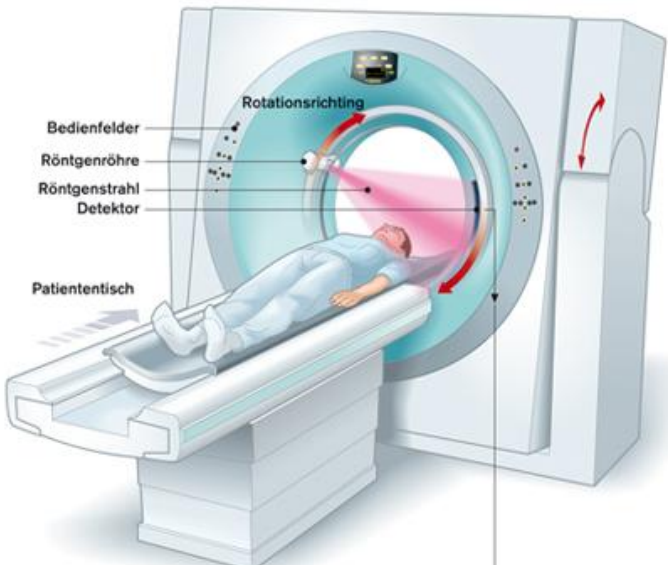

Abb. 2.2 Aufbau des CTs [Axel Kock Medienbüro für Gestaltung]

Bei einem CT Scan werden Bilder dadurch gewonnen, indem der Patient von einem Strahlenfächer durchleuchtet wird und die abgeschwächten Röntgenstrahlen von den Detektoren wieder aufgenommen werden. So ein Datensatz wird "Projektion" genannt. Werden nun mehrere Projektionen aus verschiedenen Winkeln vorgenommen, kann daraus ein Schnittbild rekonstruiert werden, dazu dreht sich der Röntgenapparat um den Körper herum. Aus mehreren hintereinander aufgenommenen Schnittbildern kann nun ein Volumsdatensatz (3D Darstellung) erstellt werden.

Bei den ersten Generationen der Computertomographen mussten die Röhre und die Detektoren bzw. der Detektor noch parallel zueinander laufen (sog. Parallelstrahl-CT-Scanner), was jedoch in einer sehr langwierigen Prozedur ausartete.

Später wurden rund um die Röhre Detektoren angebracht (sog. Detektorenring), welche abwechselt aktiviert und wieder deaktiviert werden und somit auch die Röntgenstrahlen einlesen können. Bei diesem Verfahren fällt ein Ausfall von

einem Detektor kaum mehr ins Gewicht, während bei einem Parallel Scanner ein Ausfall des Detektors enorme Folgen hatte (da sich ja immer derselbe Detektor mit dem Röntgenapparat mit drehte).

Gängige CT Typen

Die fünfte und bisher letzte Generation bilden die Spiral-CT, Elektronenstrahl CT und Kegelstrahl CT.

Beim **Spiral CT** wird wieder an das frühere Prinzip angeknüpft, in dem sich die Detektoren und der Röntgenapparat um den Körper herum drehen. In diesem Fall kann das System jedoch 360 Grad um den Körper gedreht werden und nicht mehr, wie früher, nur 180 Grad. Dadurch ist es möglich, den Patienten in konstanter Geschwindigkeit durch den Scanner zu bewegen und somit wird die Scan-Zeit deutlich verkürzt.

Beim **Elektronenstrahl CT** sind weder die Detektoren, noch die Strahlenquelle dynamisch, denn die Detektoren sind ringförmig um den Patienten angeordnet, während die Elektrodenquelle am Kopf des Gerätes fix befestigt ist. Durch eine Ablenkeinheit können die Röntgenstrahlen nun zur richtigen Position am Körper geführt werden. Aufgrund des Entfalls der drehenden Strahlenquelle ist der Elektronenstrahl CT nun viel schneller, als seine Vorgänger. Dies ermöglicht erstmals eine vernünftige Aufnahme des Herzens (innerhalb von 50 msec).

Die **Kegelstrahl CT** (auch Multislice CT genannt) verwirft nun das Konzept des Fächerstrahles und benutzt stattdessen kegelförmige Strahlen. Dadurch können, wie der englische Ausdruck "multislice" bereits verrät, mehrere Schichten auf einmal eingescannt werden. Der Vorteil dabei ist, dass der ohnehin bereits kegelförmige Strahlenbereich besser ausgenutzt wird [MUCH, Julian, 2005].

Einen Genaueren Einblick über die Technologie hinter dem CT bietet die Ausarbeitung "Computertomographie" von Julian Much.

2.3.3. Bildaufbau

Das Ziel der Computertomographie ist es, Schichtbilder zu erzeugen, bei denen jedes einzelne Pixel einen Gewebeteil repräsentiert. Einer Schicht kann eine Dicke zugeordnet werden, dadurch werden aus den Pixeln sogenannte Voxel (siehe **2.6**). Bei modernen CTs kann die Schichtdicke zwischen 0,3mm und 12mm betragen. Alles unter 0,3 mm führt beim derzeitigen Stand der Technik noch zu einer zu hohen Strahlenbelastung.

Zu jedem einzelnen Pixel soll nun der Röntgenschwächungskoeffizient (ein Mathematischer Wert, der angibt, um wie viel der Röntgenstrahl an der gegebenen Stelle abgeschwächt wurde) bestimmt werden. Dies gelingt durch Rekonstruktionsmethoden, welche über die Jahre der Forschung hinweg geändert wurden. Bei den ersten CT Apparaten wurde dies durch eine iterative

Rekonstruktionsmethode gelöst, in der die Probe in Quadrate unterteilt wurde und der jeweilige Koeffizient nacheinander berechnet wurde.

Später wurde die iterative Rekonstruktion durch die Radon-Transformation abgelöst, welche ausnutzt, dass jede beliebige, integrierbare Funktion durch alle geraden Linienintegrale über ihr Definitionsgebiet beschrieben werden kann.

Wenn nun der Röntgenschwächungskoeffizient für jedes Pixel berechnet ist, können diese in charakteristischen Grauwerten dargestellt werden. Dies erfolgt mittels der Hounsfield-Skala. Die Hounsfield-Skala wurde von G.N. Hounsfield im Jahre 1970 erfunden und repräsentiert den Dichtewert der einzelnen Volumenelemente. Den Wert 0 und somit Referenz für die anderen Werte ist Wasser, währen Luft den Wert -1000 und Knochen den Wert +3000 haben.

Da das menschliche Auge nicht mehr als 30 Grauwerte unterscheiden kann, kommt die sogenannte Fenstertechnik (siehe **Abb. 2.3**) zum Einsatz. Hierbei wird ein Fenster definiert, in dem sich die für die Behandlung interessanten Gewebsarten befinden (z.B. nur Weichteile), alles, was unter dem Fenster liegt, wird schwarz dargestellt, alles darüber weiß [BONN, Matthias, 1999-2000].

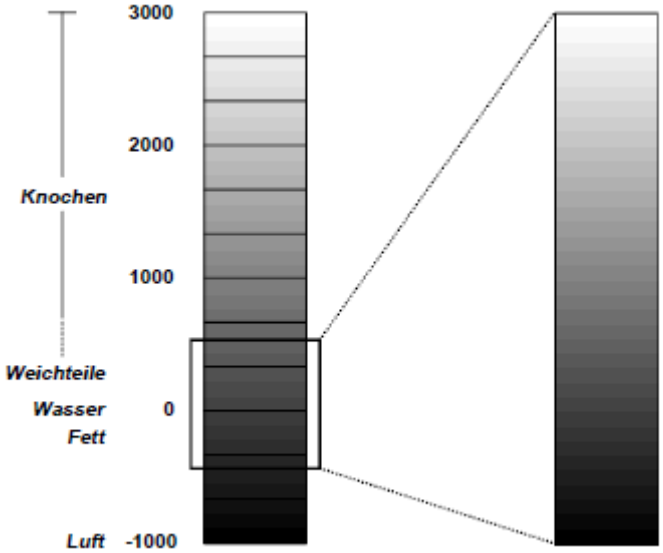

Abb. 2.3 Fenstertechnik der Hounsfield-Skala

2.3.4. Bildbeurteilung

Die Bildqualität hängt vor allem vom Auflösungsvermögen und den Artefakten im Bild ab.

Das Auflösungsvermögen ist durch die endliche Größe der Detektoren und des Fokus der Röntgenröhre begrenzt und lässt somit einen nicht viel kleineren Wert als 0,5mm zu. Diese Tatsache begrenzt die Qualität des Bildes, die dem behandelnden Arzt vorgelegt werden kann, ein.

Ein weiteres Problem sind die sogenannten Artefakte. Ein Artefakt ist eine künstliche Struktur in einem Bild, das aus verschiedenen Gründen auftreten

kann. Die Gefahr dabei ist, dass diese künstlichen Strukturen nicht unbedingt als diese ersichtlich sein müssen und dadurch Fehlinterpretationen möglich sind.

Die häufigste Art von Artefakten sind Bewegungsartefakte, die durch Bewegungen des Patienten entstehen. Diese äußern sich in Form von Streifen auf dem Bild.

Des Weiteren können sogenannte Partialvolumenartefakte (auch Teilvolumenartefakte genannt) auftreten, indem sich innerhalb eines Volumenelements mehrere Strukturen mit stark unterschiedlichem Koeffizienten befinden. Bei sehr feinen Strukturen mit scharfen Übergängen (z.B. Blutgefäße) stimmen dadurch die Dichtewerte nicht, dadurch kommt es zu Darstellungsungenauigkeiten. Diese Artefakte können nur durch eine genauere Abtastrate und eine geringere Schichtdicke verhindert bzw. vermindert werden [MUCH, Julian, 2005] [BONN, Matthias, 1999-2000] [POSPICH, Sabrina, 2012-2013].

2.4. MRT

2.4.1. Allgemein

Die Magnetresonanz-Tomographie ist ein bildgebendes Verfahren zur Darstellung von Strukturen im Inneren des Körpers. Sie kann Schnittbilder des menschlichen Körpers in beliebigen Ebenen erzeugen. Aus den Daten können mithilfe des Computers 3D-Datensätze berechnet werden. Im Gegensatz zur Computer-Tomographie hat die MRT eine bessere Weichteildarstellung.

Die erste Abbildung eines Tumors bei einem Tier entstand 1974 durch Raymond Damadian. 10 Jahre später war die MRT schließlich Verfügbar für den praktischen Einsatz.

Wie sehr sich die MRT entwickelt hat, lässt sich anhand der benötigten Zeit befestigen. Zum Beispiel brauchte man im Jahr 1980 etwa fünf Minuten für ein Schichtbild während im Jahr 2011 nur noch 100ms pro Schichtbild benötigt wurden [www.idir.uniklinikum-jena.de].

Die MRT nutzt die Kombination aus einem Magnetfeld und Hochfrequenz-Impulsen zur Anregung von Wasserstoffprotonen und nicht, wie beim CT, mit ionisierenden Strahlen. Bilder die daraus entstehen spiegeln daher nicht wie beim Röntgen die allgemein Gewebedichte wieder, sondern die Protonendichte im Gewebe. Bekannt ist die MRT auch unter dem Namen Kernspintomographie oder unter der Abkürzung MRI für die englische Übersetzung – Magnetic Resonance Imaging [PABST, Dr. med. Christoph, 2013].

2.4.2. Kernspin

Der Eigendrehimpuls eines Protons (aber nicht des ganzen Atoms) um seinen eigenen Schwerpunkt, bezeichnet man **Kernspin**. Die Elektronen des Atoms werden dabei außer Acht gelassen. Wenn z.B. ein Wasserstoff-Proton eine

positive Ladung besitzt und bewegte Ladungen ein magnetisches Moment haben, entsteht ein messbares, magnetisches Feld.

Die Eigendrehung ist eine Grundeigenschaft von Elementarteilchen, jedes Element hat dabei eine typische Frequenz, die nicht abgebremst oder beschleunigt werden kann und einfach immer da ist.

2.4.3. Funktionsweise

Um die Funktionsweise nachvollziehen zu können werden hier vorerst die Komponenten aufgelistet und dargestellt (siehe **Abb. 2.4**).

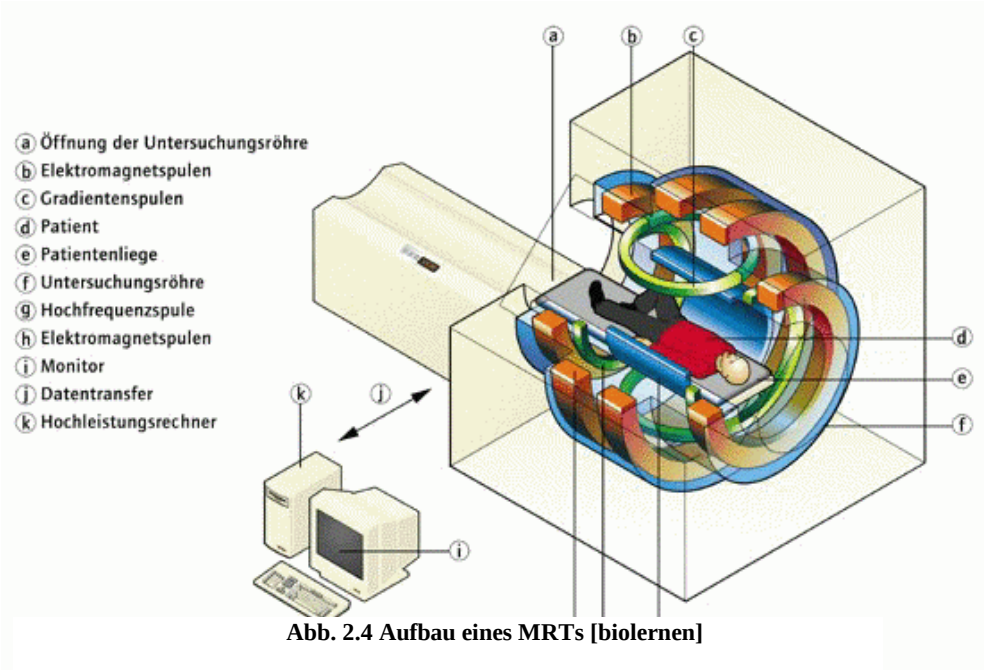

Abb. 2.4 Aufbau eines MRTs [biolernen]

Die Magnetresonanz-Tomographie nutzt den sogenannten Kernspin von Atomen (Wasserstoff, Phosphor, Kohlenstoff), die man sich vereinfacht als sehr kleine Magneten vorstellen kann, die im Menschen in großer Zahl vorhanden sind. Im Normalfall sind diese Kernspins in alle möglichen Richtungen magnetisch ausgerichtet. Gerät man in ein sehr starkes Magnetfeld, wie es beispielsweise im Inneren des MR-Tomographen der Fall ist, richten sich die Atome in dessen Richtung ähnlich einer Kompassnadel aus, was aber für den Menschen nicht spürbar ist. Der überwiegende Teil der Atome richtet sich genau mit dem Feld aus, ein verschwindend kleiner Teil genau entgegengesetzt (um 180° gedreht).

Der zu untersuchende Bereich wird nach der magnetischen Ausrichtung mit Radiowellen angeregt und die vom Körper zurückgesandten Signale werden aufgefangen. Auch dies ist nicht schädlich für den Körper. Die Frequenz der Radiowellen ist von der Magnetfeldstärke und der Kernsorte abhängig. Bei der klassischen MR-Bildgebung werden Wasserstoffatome gemessen, wobei die Radiowellen bei 1 Tesla Magnetfeldstärke eine Frequenz von etwa 42MHz haben müssen. Bei 7 Tesla arbeitet man somit mit rund 300MHz [meduniewien].

2.4.4. Bildaufbau

Um den Menschen Schicht für Schicht erfassen zu können, wird mittels zusätzlicher Magnetspulen über die zu messende Strecke ein sukzessiv ansteigend starkes Magnetfeld erzeugt, wodurch sich die Bereiche genau abgrenzen lassen. Schritt für Schritt wird nun ein Schnittbild nach dem anderen erstellt und schlussendlich setzt man alle Bilder zusammen, sodass ein dreidimensionales Bild entsteht.

Da unterschiedliches Gewebe auch einen unterschiedlich hohen Wasserstoffatom-Anteil aufweist kann man die einzelnen Bereiche sehr gut unterscheiden. Je höher die räumliche Auflösung dabei ist, abhängig von der maximalen Feldstärke, desto feiner kann man die Organe darstellen. Bei hohem Gehalt an Wasserstoffatomen im Gewebe wird der Bereich im MRT-Bild heller.

Die Kontrastauflösung der Magnetresonanz-Tomographie ist 10 mal größer als bei der Computer-Tomographie. Die Voxelauflösung bzw. die Pixelauflösung ist aber bei beiden Verfahren gleich.

2.4.5. Bildbeurteilung

Die Bildinterpretation stützt sich auf den Gesamtkontrast, die jeweilige *Gewichtung* (synonym *Wichtung*) der Messsequenz, und die Signalunterschiede zwischen bekannten und unbekannten Geweben. Im Befund wird deshalb bei der Beschreibung eines MRT-Bildes nicht von „hell“ oder „dunkel“ gesprochen, sondern von „*hyperintens“* für signalreich, hell und von „*hypointens“* für signalarm, dunkel.

Es wurden zahlreiche Messverfahren für die Untersuchung entwickelt. Zwei dieser Verfahren werden hier nun beispielhaft erklärt.

MR-Myelographie

Hierbei geht es darum, die Rückenmarksflüssigkeit ohne Kontrastmittel selektiv darzustellen. Verwendet wird dies zur Abbildung des Verlaufs der Nervenwurzeln, für die Diagnostik von Engstellen des Rückenmarkkanals oder der Zwischenwirbellöcher, sowie Zystenbildungen oder Flüssigkeitsräume.

MR-Urographie

Die MRU wird dazu verwendet, das Nierenbecken, den Harnleiter sowie Harnblase ohne Kontrastmittelgabe zu visualisieren. Angewandt wird dieses Verfahren oft zur Abklärung von entzündlichen bzw. tumorösen Veränderungen oder zur Darstellung des Nierenhohlraumsystems, der ableitenden Harnwege und Abflussstörungen [PABST, Dr. med. Christoph, 2013].

2.5. Gegenüberstellung CT – MRT

Die konkreten Unterschiede sowie die Vor- und Nachteile von der CT und MRT werden im Folgenden behandelt. Es sollte jedoch nicht als Konkurrenzkampf

angesehen werden, da sich beide Methoden ergänzen. Je nach zu untersuchender Region, diagnostischer Fragestellung, persönlicher Situation des Patienten etc. entscheiden der behandelnde Arzt und Radiologe in jedem Einzelfall, welche Diagnose-Methode sich am besten eignet.

Die MRT arbeitet mit Magnetfeldern und Radiowellen, die nicht nachgewiesen schädlich sind, während die CT mit Röntgenstrahlen arbeitet, welche bekanntermaßen den Gesundheitszustand beeinträchtigen. [radiologicum münchen]

Wann CT?

Eine CT-Untersuchung ist notwendig, wenn rasch eine zuverlässige Aussage über akute Verletzungen getroffen werden muss. Besonders bei einem Schlaganfall und SHT (Schädel-Hirn-Trauma) wird die CT bevorzugt. Auch bei der hochauflösenden Darstellung von intrapulmonaler (innerhalb des Lungengewebes liegend) und knöcherner Strukturen sowie von Nasennebenhöhlen wird die CT vermehrt genutzt. Ein Ausschlusskriterium für MRT ist z.B. das Tragen von magnetischen Fremdkörpern wie ein Herzschrittmacher oder Metallimplantate, woraufhin in den meisten Fällen auf die CT zurückgegriffen wird.

Wann MRT?

Bei einer Allergie gegen jodhaltige Röntgen-Kontrastmittel wie es bei der CT der Fall ist, oder zur Untersuchung von Kindern und jungen Erwachsenen wird großteils die MRT verwendet. Sollen beispielsweise ein Gelenk, eine Wirbelsäule, insbesondere die Bandscheiben, ein Gehirn oder generell die Weichteile dargestellt werden, wird ebenfalls die MRT angewandt [RADIOLOGIE, Gemeinschaftspraxis für].

2.6. Voxel

Der Begriff Voxel setzt sich zusammen aus „**Vo**lumen“ und „Pi**xel**“, und stellt somit einen Gitterpunkt in einem dreidimensionalen Gitter dar. Man kann also sagen, ein Voxel in einem 3D Raum ist äquivalent zum Pixel im 2D Feld. Die Position eines solchen Gitterpunkts wird nicht explizit gespeichert, sondern ergibt sich implizit aus der Position zu anderen. In **Abb. 2.5** kann man eine Menge von Voxeln sehen, wobei ein einziges davon grau markiert ist.

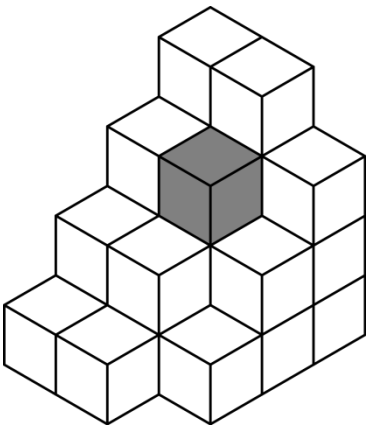

Abb. 2.5 Voxelelementmenge

Stellt ein Voxel den diskreten Wert an einer XYZ-Koordinate eines Datensatzes dar, so hat er keine bestimmte Form oder Richtung, sondern umfasst eine Menge von Daten, wofür der Begriff „isotroper Voxel“, verwendet wird. Isotrop bedeutet auch, dass sich die X, Y und Z Dimensionen in derselben Ausprägung ausweiten.

In der Medizin wird dies verwendet, um diskrete Werte wie die Dichte bei Knochen, aber auch Fettgewebe zu erkennen und anschließend zu visualisieren [de.wikipedia.org] [VOGEL, Thomas et al., 2011] [www.fallsammlung-radiologie.de].

2.7. Bildverarbeitung

Die Bildverarbeitung ist die Entwicklung von Algorithmen, die aus Bildern oder Messwerten neue Bilder erzeugen.

In der Medizin wird Bildverarbeitung eingesetzt, um die Diagnostik und die Therapie zu verbessern und dadurch pathologische Prozesse (Krankheitsprozesse) zu erkennen.

Nicht zu verwechseln ist die Bild**ver**arbeitung mit der Bild**be**arbeitung: Die Bildbearbeitung ist lediglich die Manipulation von Bildern zur anschließenden Darstellung (z.B. Photoshop) [BUCHBERGER, Michael, Zwettler, Gerald, 2014].

Die Bildverarbeitung läuft immer nach dem gleichen Muster ab, das sich "Bildverarbeitungskette" nennt.

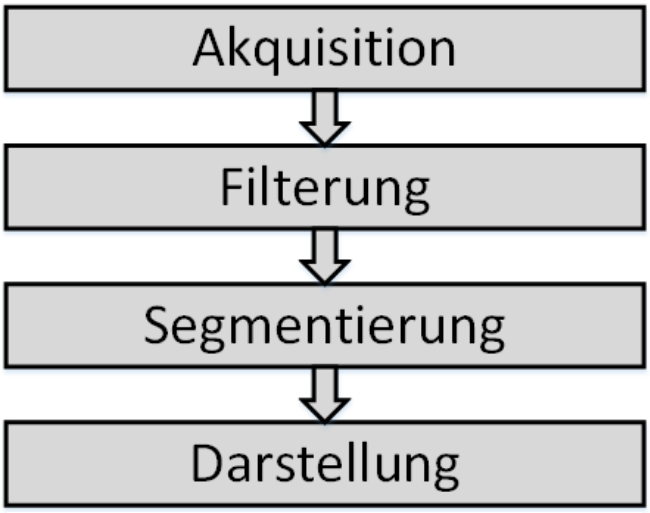

Abb. 2.6 Bildverarbeitungskette

Wie in **Abb. 2.6** zu sehen, beginnt die Kette mit der Akquisition. Hier werden die Bilddaten aufgenommen. In der Medizin müssen die Parameter bei der Aufnahme beachtet werden, wie z.B. Kontrastmittel oder Aufnahmewinkel.

Danach kommt es zur Filterung der Bilder. Dabei werden Artefakte beseitigt, geglättet und andere Filter, wie in **Kapitel 2.8** beschrieben, angewandt.

Als Nächstes wird das Bild segmentiert. Dabei werden relevante und nicht relevante Teile des Bildes getrennt. Mehr dazu ist in **Kapitel 2.9** zu finden.

Zum Schluss wird das gefilterte und segmentierte Bild dargestellt. Dies kann in 2D oder auch in 3D erfolgen. Dabei können auch Prozesse wie "Direct Volume Rendering" genutzt werden, um Bereiche farblich hervorzuheben, beziehungsweise die Transparenz des Bildes einzustellen [ZWETTLER, Gerald, 2013].

2.8. Grafikfilter

Soll ein Bild verbessert werden, um z.B. bestimmte Objekte besser erkennen zu können, so muss ein Filter angewandt werden. Das heißt es können Inhalte aus einem Bild hervorgehoben und Störungen beseitigt werden. Dabei wird immer versucht, keine wichtigen Informationen zu verlieren.

2.8.1. Lineare Filter

Lineare Filter verwenden eine Maske mit Gewichtsfaktoren, womit zu jedem Bildpunkt eine Linearkombination aus dem Gewichtsfaktor und den Pixelwerten in der Umgebung des ausgewählten Pixels errechnet wird. Bleibt die Filtermaske für jeden Bildpunkt gleich, so spricht man von einem homogenen Filter. Beispiele für lineare Filter wäre der Mittelwertfilter, Tief- und Hochpassfilter und der Wiener Filter.

2.8.2. Nichtlineare Filter

Diese ermöglichen im Gegensatz zu linearen Filtern meist eine deutlich gesteigerte Bildqualität beim Verarbeiten. Beispiele hierfür wäre der Medianfilter, morphologische Filter oder der anistropische adaptive Filter.

2.8.3. Adaptive Filter

Das Ergebnis des adaptiven Filters hängt stark von der Umgebung der Pixel ab. Sie können das Rauschen in Bildern verringern, ohne dabei andere Bildinformationen zu verwischen. So bleiben z.B. Kanten im Bild erhalten [PAULUS, Dietrich, Bouattour, Sahla, 2003].

2.9. Segmentierung

Die Segmentierung ist ein Teilgebiet der digitalen Bildverarbeitung. Das Ziel dabei ist es, ein Bild in relevante und nicht relevante Regionen durch Zusammenfassung von Pixel oder Voxel zu unterteilen.

Dazu wird für jeden Bildpunkt über entsprechende Methoden entschieden, zu welchem Objekt er gehört.

Als Vorverarbeitung ist es empfehlenswert das Bild in ein Graustufenbild umzuwandeln. Es sollten Fehler, die über Rauschquellen entstehen herausgefiltert und unterschiedliche Beleuchtungen ausgeglichen werden.

Konkret in MeVisLab wird Segmentierung benötigt, um gewisse Teile eines medizinischen Bildes explizit hervorzuheben, da so etwaige Probleme besser erkannt werden können.

Folgende Methoden der Segmentierung existieren:

2.9.1. Pixelbasierte Methode

Arbeitet mit den Grauwerten der Pixel, bezieht aber die Grauwerte der Nachbarpixel nicht mit ein. Bei der Pixelbasierten Methode wird für jeden einzelnen Bildpunkt eine Entscheidung getroffen, ob dieser in das Segment gehört oder nicht. Dies wird mithilfe eines Schwellwertes festgelegt, der überschritten werden muss, damit die Bedingung eintrifft. Dieser Schwellwert kann der Grauwert eines Pixels sein.

Wenn nun das Schwellwertverfahren angewandt wird, wird jeder Bildpunkt mit dem Schwellwert verglichen und je nach Ausgang des Vergleichs wird der Bildpunkt im Zielbild auf "1" oder "0" gesetzt. Dadurch entsteht ein sogenanntes Binärbild (Ein Bild, das nur die Farben Schwarz oder Weiß annehmen kann).

Wie bereits erwähnt, können Bildrauschen und Helligkeitsunterschiede zu Problemen führen, daher sollten diese Probleme vorher schon behandelt werden.

Besonders geeignet ist das Schwellwertverfahren in der Texterkennung in Bildern [PETERWITZ, Julia, 2006] [BORGHOFF, Thorsten, Höll, Nico, 2005].

2.9.2. Modellbasierte Methoden

Die modellbasierte Segmentierung versucht über bestimmte Formen zu segmentieren, dies erfordert vorher eine Kantendetektion (siehe **Abb. 2.8** und Abb. 2.7**)**. Bei der Modellbasierten Methode wird ein Modell des gesuchten Objektes zugrunde gelegt. Das kann zum Beispiel eine Form sein, man weiß also schon vorher, was man ungefähr sucht.

Abb. 2.8 Originalbild

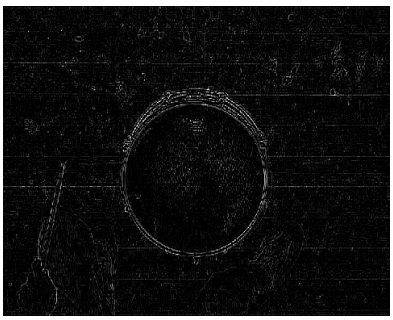

Abb. 2.7 Kantendetektion

Ein bekanntes Verfahren zur Erkennung von Geraden, Kreisen oder anderen parametrisierbaren geometrischen Figuren ist die Hough-Transformation. Um die Form zu erkennen, ist es nötig, vorher eine Kantenerkennung durchzuführen und das daraus resultierende Schwarz-Weiß-Bild zu verwenden. Werden nun die nötigen Parameter für die Hough-Transformation eingegeben, beispielsweise die x und y Koordinaten des Mittelpunkt eines Kreises, kann diese starten und die gewünschte Form finden. Diese Methode wird z.B. für Gesichtserkennung oder zur Erkennung von Fahrbahnlinien verwendet [PETERWITZ, Julia, 2006] [BORGHOFF, Thorsten, Höll, Nico, 2005].

2.9.3. Kantenbasierten Methoden

In der Kantensegmentierung wird nach Kanten bzw. Objektübergängen in einem Bild gesucht. Das Ergebnis dieser Methode sind zum Beispiel Polygone oder Linien. Des Weiteren soll ein homogener Bereich erkannt werden (also ein Bereich mit ähnlichen bzw. gleichen Eigenschaften wie z.B. die Farbe).

Das Problem hierbei ist wieder die Qualität der Bilder, da meist die Kanten nicht vollständig zusammenhängen und somit auch das Ergebnis der Kantensegmentierung darunter leidet. Daher ist es, wie bei der Pixelsegmentierung, notwendig, die Bildqualität vorher zu verbessern bzw. die Kanten auszubessern. Dies ist durch den "Sobel-Operator" und dem "Laplace-Operator" möglich. Die Kantendetektion eignet sich dafür, ein Bild in eine Vordergrundebene und in eine Hintergrundebene zu unterteilen. Im speziellen Fall von medizinischen Bildern ist die Kantendetektion z.B. interessant für die Detektion von Tumoren [PETERWITZ, Julia, 2006] [BORGHOFF, Thorsten, Höll, Nico, 2005].

2.9.4. Regionsbasierte Verfahren

Regionsbasierte Verfahren untersuchen die Grauwerte in zusammenhängenden Regionen. Diese Verfahren analysieren lediglich homogene Bereiche. Im Gegensatz zur pixelorientierten Methode betrachten die regionsorientierten Verfahren Punktmengen als Gesamtheit und versuchen dadurch zusammenhängende Objekte zu finden. Ein bekanntes Verfahren zur Verschmelzung von Bildbereichen ist das Region Growing.

Zu Beginn wird das Bild in Zellen unterteilt. Dann wird eine initiale Zelle betrachtet und als Startwert angenommen. Nun werden alle Nachbarn auf die Eigenschaften (z.B. Mittelwert der Farbe) abgetestet. Liegen gleiche oder ähnliche Werte vor, werden die Nachbarn zur Region hinzugefügt und somit "verschmolzen". Werden keine Nachbarn mehr gefunden, die der Region hinzugefügt werden können, wird eine neue Zelle gewählt, die nicht zur Region gehört, und die ganze Prozedur beginnt wieder von vorne, bis kein unberührtes Pixel mehr vorhanden ist.

Dieses Verfahren eignet sich besonders gut für die medizinische Bildverarbeitung, weil dadurch zusammengehörende Bereiche herausgefiltert werden. Das kann ein Organ wie eine Niere sein, aber auch zum Beispiel eine von Medizinern eingesetzte Schraube. [PETERWITZ, Julia, 2006] [BORGHOFF, Thorsten, Höll, Nico, 2005].

2.9.5. Texturbasierte Methoden

Das Erkennen von bestimmten Texturen im Bild wird mithilfe von texturbasierten Methoden erreicht. Manche Bilder besitzen keine einheitlichen Farben, sondern einheitliche Texturen, wie zum Beispiel Rillen, die auf dem Foto als helle und dunkle Streifen erscheinen. Eine Textur definiert die Oberflächenbeschaffenheit von Objekten [PETERWITZ, Julia, 2006] [BORGHOFF, Thorsten, Höll, Nico, 2005].

2.10. DICOM Standard

Digital Imaging and Communications in Medicine (DICOM) ist der Standard für die Kommunikation und Verwaltung von medizinischen Auswertungen und den zugrundeliegenden Bilddaten. Bei der Einführung der Computertomographie, gefolgt von anderen digitalen Bildverarbeitungen zu Diagnosezwecken im Jahre 1970, erkannte die ACR (American College of Radiology) und die NEMA (National Electrical Manufacturers Association), dass es notwendig sein wird, einen Standard für die Datenübertragung von Bildern und zusätzlichen Informationen zu definieren. Diese Organisationen schlossen sich schließlich 1983 zusammen um DICOM zu entwickeln [NEMA, 2015].

DICOM enthält eine Vielzahl von Funktionalitäten wie z.B. netzwerkorientierte Dienste, die die Bildübertragung, das Drucken oder die Abfrage eines Bildarchivs ermöglichen. Weiters werden Datenstrukturen und Formate für medizinische Bilder und bildbezogene Daten definiert.

Die von DICOM angebotenen **Netzwerkdienste** arbeiten nach dem typischen Client-Server Konzept. Um Daten auszutauschen, muss vorher eine Verbindung aufgebaut werden. Damit diese zustande kommt, muss ausgehandelt werden, wer der Server (Serviceclass Provider) und wer der Client (Serviceclass User) ist, welche DICOM-Dienste überhaupt in Anspruch genommen werden und in welchem Format und in welcher Kompression die Daten übertragen werden.

Einige Beispiele zu Netzwerkdiensten:

"*Store*" empfängt Daten und versucht diese abzuspeichern.
"*Query/Retrieve*" erhält Datenbankanfragen und sendet das Ergebnis zurück an den Client.
"*Print Management*" druckt empfangene Daten.
"*Worklist Management*" beschäftigt sich mit der Verwaltung der einzelnen Arbeitsschritte, die für einen Patienten notwendig sind.

Die **Datenstruktur** eines DICOM-Bildes besteht aus vielen Datenelementen- bzw. –attributen, die bildbegleitende Informationen enthalten. Hierbei gibt es natürlich verschiedene Arten von Informationen wie z.B. den Patienten betreffend, der Modalität bzw. Aufnahme betreffend oder auch generelle Bildinformationen wie die Auflösung oder Fensterung [dicom.offis.de, 2013].

3. Technische Grundlagen

In diesem Kapitel werden wichtige Themen der Informatik erklärt.

3.1. Arbeitsumgebungen

Dieser Punkt behandelt die verschiedenen Softwareprodukte, die bei der Diplomarbeit angewandt wurden.

3.1.1. MeVisLab

MeVisLab bietet eine Plattform für die Bildverarbeitung, für Forschung und Entwicklung mit dem Schwerpunkt der medizinischen Bildgebung. Es ermöglicht eine schnelle Integration und Erprobung von neuen Algorithmen und die Entwicklung von Anwendungsprototypen, die in klinischen Umgebungen verwendet werden können.

Außerdem enthält es fortschrittliche medizinische Module zur Segmentierung, Registrierung, Volumetrie und quantitativen morphologischen (=Aufbau/Struktur von Organismen und Zellen) und funktionellen Analyse. Mehrere klinische Prototypen wurden aufgrund von MeVisLab realisiert, einschließlich Software-Assistenten zur bildgebenden, dynamischen Bildanalyse, zur chirurgischen Planung und zur Gefäßanalyse.

Die Umsetzung nutzt eine Reihe von bekannten Bibliotheken von Drittanbietern und Technologien, vor allem die Anwendungs-Frameworks Qt, das Visualisierungs- und Interaktions-Toolkit Open Inventor, die Skriptsprache Python, und den Grafikstandard OpenGL [www.mevislab.de].

3.1.2. Microsoft Visual Studio 2013

Microsoft Visual Studio 2013 ist eine Entwicklungsumgebung zur einfachen Bereitstellung von Anwendungen auf allen Microsoft Plattformen. Agile Softwareentwicklung inklusive Sprintplanung und Burndowndiagramme können erarbeitet werden. Das Zusammenarbeiten im Team ist auch kein Problem, denn mit Git-basierten Repositorys kann jeder gleichzeitig am Projekt arbeiten. Wie in jeder ordentlichen Entwicklungsumgebung ermöglicht auch Visual Studio das Debuggen eines Programmes sowie das Modellieren von UML Diagrammen [www.visualstudio.com, 2015].

3.2. MDL

MeVisLab Definition Language ist eine Sprache zur Konfiguration und zur Definition von Oberflächen. Üblicherweise enden die Source Code Dateien mit „.def“ oder „.script“. MDL ist ein typisches Key=Value Konzept, das heißt einem „Tag“ wird ein bestimmter Wert zugewiesen, oder aber auch einer Menge von anderen „Tags“. Es bietet auch die Möglichkeit, externe Scripts aufzurufen, die in Programmiersprachen wie Javascript oder Python geschrieben sind [www.mevislab.de].

3.3. HTML Agility Pack

Das HTML Agility Pack ist eine .NET Bibliothek, die es ermöglicht HTML Dokumente einzulesen und daraus eine hierarchisch strukturierte Ablageform zu bilden. Die Bibliothek nimmt auch Rücksicht auf falsch formatiertes HTML. Die einzige Abhängigkeit zu anderen Frameworks bzw. Programmen besteht zu .NET und zur Implementierung des XPATH (Ermöglicht die Adressierung von Elementen in einem XML Dokument, das wie ein Baum aufgebaut ist.) [CLARK, James, DeRose, Steve, 1999].

Um die Bibliothek nutzen zu können, sollte sie vorher als Referenz im C# Projekt hinzugefügt werden. Will man beispielsweise alle Links auf einer Webseite durchlaufen und ausgeben, ist folgendes Vorgehen nötig.

```
1  var getHtmlWeb = new HtmlWeb();
2  var document = getHtmlWeb.Load(InputTextBox.Text);
3  var aTags = document.DocumentNode.SelectNodes("//a");
4  if (aTags != null)
5  {
6     foreach (var aTag in aTags)
7     {
8        Console.WriteLine(aTag.Attributes["href"].Value);
9     }
10 }
```

Codebeispiel 3.1 HTML-Agility-Pack

In **Codebeispiel 3.1**, Zeile 1, wird ein Objekt der Klasse „*HtmlWeb*“ erzeugt, damit die „*Loa*d*()*“ Methode davon benutzt werden kann.

Das Beispiel in **Codebeispiel 3.1**, Zeile 2, speichert den Rückgabewert der `Load`-Methode, in eine Variable der Klasse „*HtmlDocument*“, um auf die einzelnen DOM-Elemente zugreifen zu können (Ein DOM-Element entspricht einem Knoten in einem hierarchisch aufgebauten Dokument).
HTML-Links entsprechen großteils dem Schema in **Codebeispiel 3.2**.

```
1  <a href=www.link.com>Text</a>
```

Codebeispiel 3.2 Link in HTML Syntax

In Zeile 3 vom **Codebeispiel 3.1Fehler! Verweisquelle konnte nicht gefunden werden.** wird mit der Methode „*SelectNodes()*“ auf diese Links explizit zugegriffen. Anschließend wird in der Variable „*aTags*“ nun eine Liste von DOM-Elementen, die ein <*a*> Tag enthalten, gespeichert. Zeilen 4 bis 9 zeigen das Vorgehen für das Ausgeben aller Elemente mittels einer foreach-Schleife. Um die Linkadresse aus dem DOM-Element herauszuholen, werden die Attribute nach dem Stichwort „*href*“ gesucht, damit die entsprechende Linkadresse durch Zugriff auf den „*value*“ (siehe Zeile 8). Wie man sieht erfolgt

der Zugriff auf Attribute mithilfe einer Hashmap, die als Schlüssel und Wert den Datentyp „String“ verwendet [SHANTO, Asherd, 2015] [htmlagilitypack.codeplex.com, 2012].

3.4. AForge.NET framework

Das AForge.NET Framework ist ein C# Framework, welches für Entwickler und Forscher in den Bereichen Computer Vision (Studiengang) und künstliche Intelligenz designt wurde. Es unterstützt unter anderem Bildbearbeitung, Robotik und maschinelles Lernen. Das Framework besteht aus mehreren Bibliotheken:

- AForge.Imaging - Bibliothek mit Bilbearbeitungsprozessen und Filtern
- AForge.Video - Bibliothek zur Videobearbeitung
- AForge.MachineLearning - Bibliothek für maschinelles Lernen
- etc.

Die für dieses Projekt relevante Bibliothek stellt die AForge.Imaging Bibliothek dar.

3.4.1. AForge.Imaging

Die AForge.Imaging Bibliothek (Aforge.Imaging.dll) ist die größte des Frameworks. Sie beinhaltet unterschiedliche Bildbearbeitungsroutinen, wie zum Beispiel:

- Grafikfilter
- Kantendetektion
- Größenänderung und Rotation
- Texturenfilter
- Bildvergleich
- etc.

Verwendet wurde der Punkt "Bildvergleich", genau gesagt die Klasse "*ExhaustiveTemplateMatching*".

Diese Klasse implementiert den "exhaustive template matching" (vollständiger Schablonenabgleich) Algorithmus, welcher das gesamte gewünschte Bild durchscannt und es Pixel für Pixel mit der "Schablone" (also das Bild, mit dem man es vergleichen will) vergleicht. Es ist nur möglich, 8 bpp Graustufen und 24 bpp Farbbilder zu verarbeiten. "bpp" bedeutet in diesem Zusammenhang "bits per pixel" und beschreibt die Farbtiefe, das heißt, wie fein die Abstufung zwischen den einzelnen Farben ist.

Beispiel:

```
1  ExhaustiveTemplateMatching tm =
2  new ExhaustiveTemplateMatching( 0 );
3  TemplateMatch[] matchings = tm.ProcessImage( image1,
4                                               image2 );
5  if ( matchings[0].Similarity > 0.95f )
6  {
7      // do something with quite similar images
8  }
```

Codebeispiel 3.3 ExhaustiveTemplateMatching

Wie in **Codebeispiel 3.3** zu sehen, wird ein Objekt der Klasse "ExhaustiveTemplateMatching" angelegt, welches die beiden Bilder, die in Zeile 3 eingelesen werden, vergleicht.

Als Resultat erhält man eine Fließkommazahl, dargestellt in Zeile 4, zwischen 0 und 1, welche den Ähnlichkeitsgrad angibt. 0 heißt, sie sind sich zu 0% ähnlich, 1 heißt, dass sie zu 100% das gleiche Bild sind [AFORGE.NET, 2008-2012] [KIRILLLOV, Andre].

3.5. WPF

Windows Presentation Foundation, kurz WPF, ist ein System zum Entwerfen von Windows-Clientanwendungen sowie für im Browser gehostete Anwendungen. Der Kern von WPF ist ein auflösungsunabhängiges und vektorbasiertes Renderingmodul, das die Funktionen moderner Grafikhardware nutzt. Dieser Kern wird ergänzt durch ein umfassendes Arsenal an Anwendungsentwicklungsfunktionen wie z.B. Extensible Application Markup Language – XAML – (siehe **Kapitel 3.6**), Steuerelemente, Datenbindung, Layout, 2D- und 3D Grafik und vieles mehr. Es ist möglich, die Anwendung sowohl mit XAML, als auch mit dem Code-Behind, also dem C# Code, der die Logik im Hintergrund verwaltet, zu entwickeln.

Dadurch ergibt sich eine klare Trennung zwischen **Darstellung** und **Verhalten**, was natürlich einige Vorteile mit sich bringt. Darstellungsspezifischer Markup und verhaltensspezifischer Code-Behind sind nicht eng miteinander verflochten, darum kann die Komplexität der Entwicklung eingeschränkt und die Wartungskosten verringert werden. Ein weiterer Vorteil wäre, dass Designer und Entwickler parallel ihren Anwendungsbereich bearbeiten können, ohne sich in die Quere zu kommen.

Steuerelemente spielen eine große Rolle im Bereich WPF. Damit gemeint ist ein Sammelbegriff, der eine Kategorie von WPF-Klassen umfasst, die in einem Fenster oder auf einer Webseite vorzufinden sind und über eine eigene

Benutzeroberfläche sowie ein eigenes Verhalten verfügen. Beispielsweise gibt es für die Anzeige von Daten die vordefinierten Klassen “DataGrid“, „ListView“ oder „TreeView“.

Unter **Datenbindung** versteht man grundsätzlich das Kopieren der Daten aus den verwalteten Objekten in Steuerelementen, sodass diese Daten angezeigt und bearbeitet werden können. Wichtig dabei ist, dass vorgenommene Änderungen mithilfe der Steuerelemente wieder zurück in die verwalteten Objekte geschrieben werden. In **Abb. 3.1** Databinding in .NET wird das Data-Binding einfach und verständlich dargestellt [Einführung in WPF].

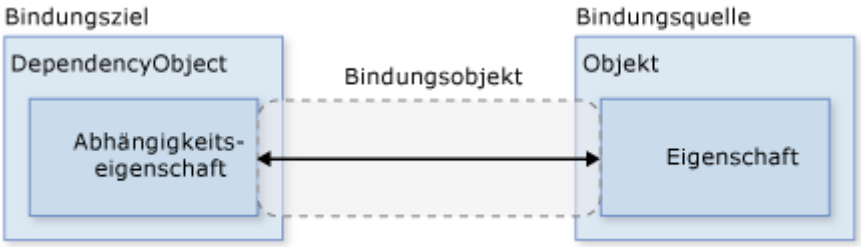

Abb. 3.1 Databinding in .NET [msdn.microsoft.com]

3.6. XAML

XAML ist eine von Microsoft entwickelte deklarative Markupsprache, die das Konstruieren einer Benutzeroberfläche von .NET Anwendungen wesentlich vereinfacht. Benutzeroberflächenelemente können entweder per Drag and Drop oder auch manuell erstellt werden. Rein optisch gesehen erinnert XAML-Code sehr an die Auszeichnungssprache XML.

Ein **Objektelement** deklariert ein Objekt eines Typs, der in den Assemblies von .NET Umgebungen definiert ist. Die Syntax eines solchen Objektelements beginnt immer mit einer spitzen Klammer, die sich öffnet „<“. Gleich darauf wird der Name des Typs angegeben. Schließlich können noch Attribute deklariert werden. Um das Element zu schließen kann nun ein Schrägstrich und eine schließende spitze Klammer „/>“ angefügt werden. Ein alternativer Weg, das Element zu schließen, wäre, nach den Attributen eine schließende Spitze Klammer „>“ anzuhängen, dann den Inhalt des Elements einzufügen und schlussendlich ein Ende-Tag (z.B. „**</Typname>**“) anzugeben.

Ein vollständiges Beispiel einer solchen Objektelementdeklaration ist in **Codebeispiel 3.4** Objektelementdeklaration angegeben.

```
1  <StackPanel>
2    <Button Content="Click Me"/>
3  </StackPanel>
```

Codebeispiel 3.4 Objektelementdeklaration

Markuperweiterungen sind ein XAML-Sprachkonzept und werden verwendet, indem man beim Festlegen eines Wertes für ein Attribut geschweifte Klammern „{„ und „}“ angibt. Bei WPF-Anwendungen werden vor allem Markuperweiterungen wie die Datenbindung, „StaticResource“ und „DynamicResource“ genutzt. Unter „StaticResource“ versteht man Ressourcen, die bereits bei der Implementierung von Anwendungen vorhanden sind, jedoch nur instanziiert werden müssen, und mit „DynamicResource“ sind Ressourcen gemeint, die dynamisch zur Laufzeit entstehen.

```
1  <Page.Resources>
2    <SolidColorBrush x:Key="MyBrush" Color="Gold"/>
3    <Style TargetType="Border" x:Key="PageBackground">
4      <Setter Property="Background" Value="Blue"/>
5    </Style>
6  </Page.Resources>
7  <StackPanel>
8    <Border Style="{StaticResource PageBackground}">
9    </Border>
10 </StackPanel>
```

Codebeispiel 3.5 Markuperweiterungen

Im Beispiel in **Codebeispiel 3.5** Markuperweiterungen, Zeile 8, enthält die Style-Eigenschaft im Objektelement „Border“ ein Objekt der Klasse „Style“. Dieses Objekt existiert nur bei Laufzeit und kann mithilfe von „StaticResource“ auf das Attribut „Style“ zugewiesen werden [Einführung in WPF].

3.7. ML Modul

Das sogenannte „ML Modul“ bezeichnet eine Bildverarbeitungsbibliothek (MeVis Image Processing Library), bietet alle möglichen Arten (etwa 300) der Bildverarbeitung an und kann bis zu 6 Dimensionen (x, y, z, Farbe, Zeit, andere Dimension) verarbeiten. Die Bibliothek ist in der Programmiersprache C++ geschrieben und kann somit auf allen gängigen Plattformen wie Windows, Linux und MAC OS X ausgeführt werden, wobei es zu beachten ist, dass auch eine C Schnittstelle zur Verfügung gestellt wird. Die Verfahren der Bildverarbeitung sowie deren Algorithmen werden als einzelne Module repräsentiert. Sie können durch eine grafische Benutzeroberfläche miteinander beliebig verbunden werden, was den Fluss der Bilddaten gut veranschaulicht [MEVIS MEDICAL SOLUTIONS, 2015] [www.mevislab.de].

4. Beschreibung der Anwendung

4.1. Installation des Moduls

Um das MeVisLab-Modul für diese Diplomarbeit einzubinden ist es notwendig den Installer (siehe **Kapitel 6.4**), der der Diplomarbeit beiliegt, auszuführen. Außerdem muss im Programm **MeVisLab Debug** im Reiter „Edit“ beim Punkt „Preferences“ im Menü "Packages" das Benutzerpaket, das der Diplomarbeit beiliegt, hinzugefügt werden. Dies ist möglich, indem man den Button "Add Existing User Packages" klickt, und den Ordner des beigelegten ML-Moduls auswählt. Ganz wichtig hierbei ist, dass immer die Debug Version von MeVisLab (dies gilt auch für später beschriebene Kapitel) gestartet wird, da man für die Standardversion eine Entwicklerlizenz braucht, die das Budget sprengen würde. Wenn das Benutzerpaket hinzugefügt wurde, startet sich MeVisLab neu, jedoch ist es notwendig, das Programm ein weiteres Mal neuzustarten, da ansonsten ein Fehler seitens MeVisLab auftritt. Natürlich muss der beschriebene Vorgang zur Installation nur einmal durchgeführt werden.

4.2. Verwendung des Moduls

Um das MLA-Modul zu verwenden, muss vorher MeVisLab gestartet werden. Danach klickt man auf den Menüpunkt "Modules" → "Extras" → "Tutorials" → "MLATut". Nun öffnet sich ein Fenster namens "Panel MLATut".

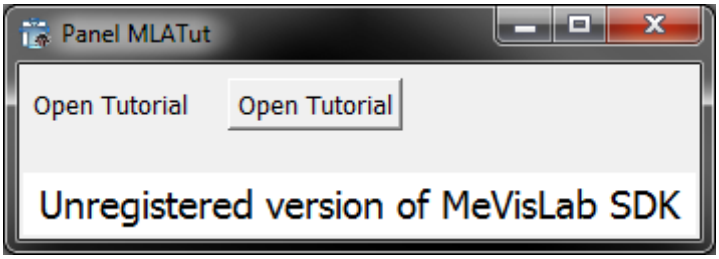

Abb. 4.1 Panel MLATut

Wie auf **Abb. 4.1** Panel MLATut zu sehen, muss nun der Button "Open Tutorial" geklickt werden, um die C#-Anwendung zu starten.

Folgende Ansicht ist nun zu sehen:

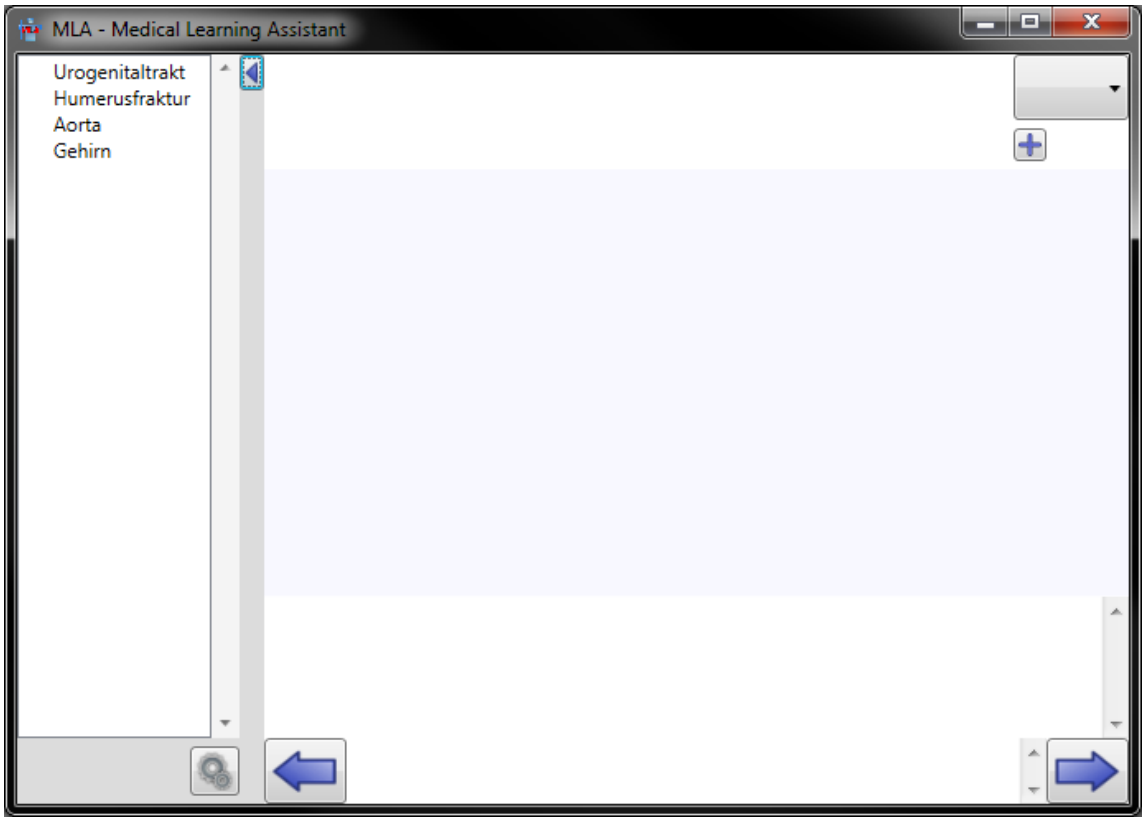

Abb. 4.2 Start der Anwendung

Wie in **Abb. 4.2** Start der Anwendung abgebildet, sieht man vorerst eine Aufzählung von bereits in HTML umgewandelten Anwendungsbeispielen. Um nun ein Tutorial zu öffnen, muss im Baum links ein Beispiel ausgewählt werden.

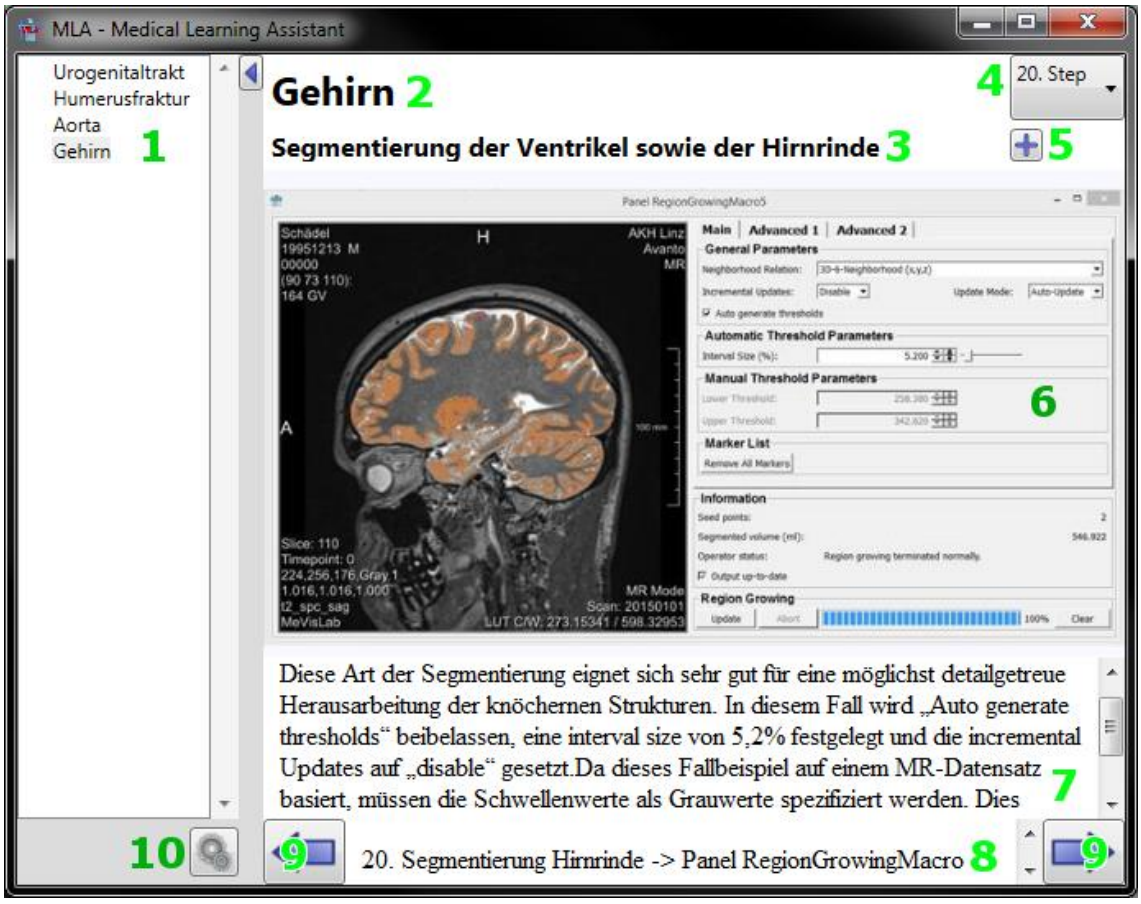

Abb. 4.3 Auswahl eines Beispiels

Die folgenden 10 Nummern beziehen sich jeweils auf die **Abb. 4.3** Auswahl eines Beispiels**.**

1) Die Anwendungsbeispiele, die im "Documents" Ordner beinhaltet sind.
2) Die Überschrift des Beispiels
3) Der Kapitelname des ausgewählten Kapitels
4) Der sogenannte "Stepselector" ist ein Dropdown, in dem man den gewünschten Schritt auswählen kann
5) Beim Klicken auf den Button mit dem "+" Symbol wird die Beschreibung des ausgewählten Kapitels angezeigt.
6) In der Mitte des Fensters befinden sich die Fotos eines selektierten Schrittes
7) Die schriftliche Anleitung eines ausgewählten Schrittes
8) Die Überschrift eines ausgewählten Schrittes
9) Navigieren zu einem vorherigen bzw. nachfolgenden Schritt
10) Die Einstellungen, hier befindet sich ebenfalls das Importmodul zum importieren neuer Tutorials

4.3. Erstellen einer eigenen Anleitung

Beim Erstellen einer eigenen Anleitung ist besonders die Formatierung zu beachten. Die Formatvorlage für eigene Dokumente liegt der Diplomarbeit bei (Formatvorlage.docx). In Kapitel **4.3.1** ist eine Anleitung beispielhaft eingebunden. Natürlich wurde nicht der gesamte Inhalt davon übernommen, sondern nur ein Teil zur Veranschaulichung.

In der Formatvorlage sind alle möglichen Fälle beschrieben. Kurz zusammengefasst ergibt sich daraus folgendes (in Klammer die Schriftart):

1x Dokumentenname erlaubt (Überschrift 1 BAC)

1x generelle Beschreibung zum Dokument erlaubt (Standard)

beliebige Anzahl an Kapiteln erlaubt (Überschrift 2 BAC)

1x Beschreibung zum Kapitel erlaubt (Standard)

innerhalb eines Kapitels beliebige Anzahl an Schritten erlaubt (Überschrift 3 BAC)

beliebig langer Text zu einem Schritt erlaubt (Standard)

innerhalb eines Schritts beliebige Anzahl an Bildern erlaubt

1x pro Bild eine Bildbeschriftung erlaubt (Standard), diese muss sofort nach dem Bild kommen, ohne jegliche Leerzeilen

Abzuspeichern ist das Dokument im Microsoft Word ".docx" Format.

4.3.1. Beispielanleitung – Gehirn

Gehirn

Dieses Fallbeispiel behandelt die Segmentierung und Zusammenfügung verschiedener Strukturen des Gehirns. Visualisiert werden die Hirnrinde, das Hirnmark und das Ventrikelsystem. Zusätzlich erfolgt die Darstellung eines anatomischen Hintergrundes.

Segmentierung des Hirnmarks und des anatomischen Hintergrundes

Im Folgenden werden die Arbeitsabläufe zur Darstellung des Hirnmarks beschrieben. Weiters werden zur anatomischen Orientierung die Gesichts- und Schädelstrukturen dargestellt.

Datensatz laden --> Panel ImageLoad

Der Datensatz wird mit dem Modul ImageLoad geöffnet.

Dateiname: MR Schädel_MPR.dcm

Segmentierung Marklager -> Panel RegionGrowingMacro

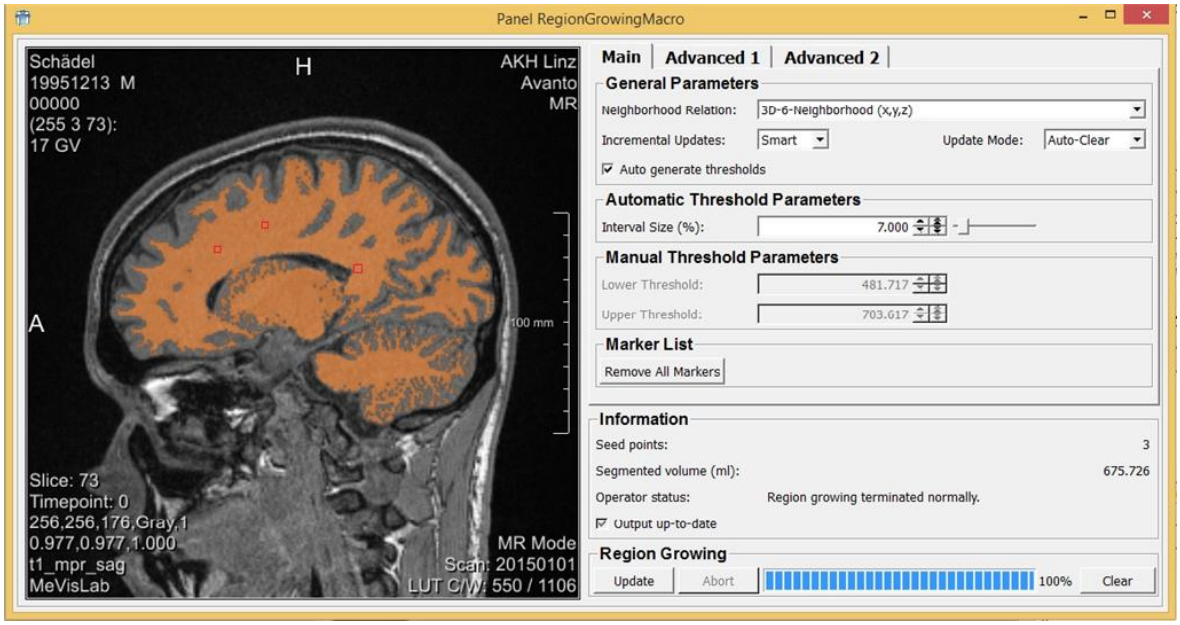

Abb. 4.4: Segmentierung des Hirnnmarks (Knogler, Panzenböck, 2015)

Diese Art der Segmentierung (siehe **Abb. 4.4**) eignet sich sehr gut für eine möglichst detailgetreue Herausarbeitung der knöchernen Strukturen. In diesem Fall wird „Auto generate thresholds“ beibelassen und eine interval size von 7% festgelegt.

Da dieses Fallbeispiel auf einem MR-Datensatz basiert, müssen die Schwellenwerte als Grauwerte spezifiziert werden. Dies erfolgt im Reiter „Advanced 1“.

Es empfiehlt sich außerdem "Persistant markers" zu aktivieren, um kein erneutes Segmentieren bei einem neuen Programmstart ausführen zu müssen.

Filterung des Datensatzes → Panel RecGaussFilter

Um eine akkurate Glättung und somit eine möglichst optimale Visualisierung des Marklagers zu erreichen, eignet sich der RecGaussFilter. Durch verstellen der Kernel in die 3 Raumachsen, erreicht man für dieses Beispiel ein gutes Ergebnis. Hierbei wird für x und y der Wert 4 und für z der Wert 3 eingestellt.

Volumenrendering → SoGVRVolumeRenderer

Dieses Modul erlaubt ein hochqualitatives 3D und 4D Volumenrendering. Zur optimalen Visualisierung sollte der „Render Mode“ auf Illuminated gesetzt werden.

Bearbeitung der LUT → SoLUTEditor

Über dieses Modul können die „RGBA LUT’s“ hinsichtlich ihrer Farbe und Transparenz modifiziert werden. Die folgende Abbildung zeigt die Einstellung für dieses Beispiel.

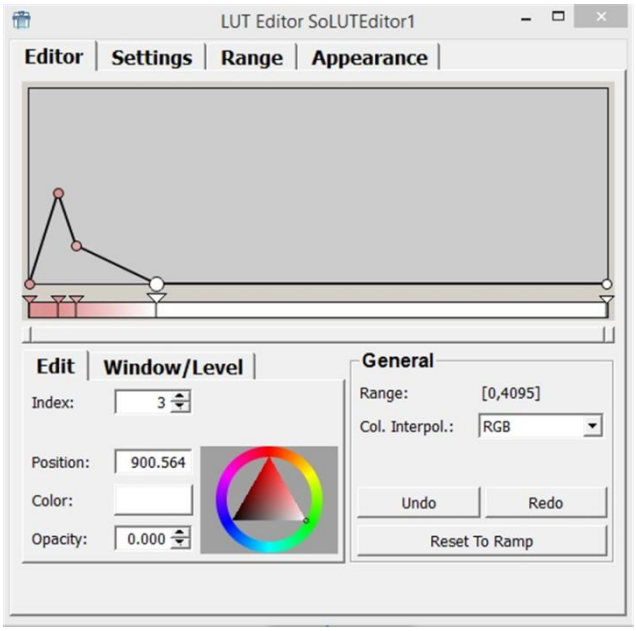

Abb. 4.5: Bearbeitung der LUT des Hirnmarks (Knogler, Panzenböck, 2015)

CAVE: Es ist erforderlich das SoLUTEditor-Modul mit dem linken Konnektorfeld des folgenden SoGroup-Moduls zu verbinden, da der Editor sonst nicht aktiv ist.

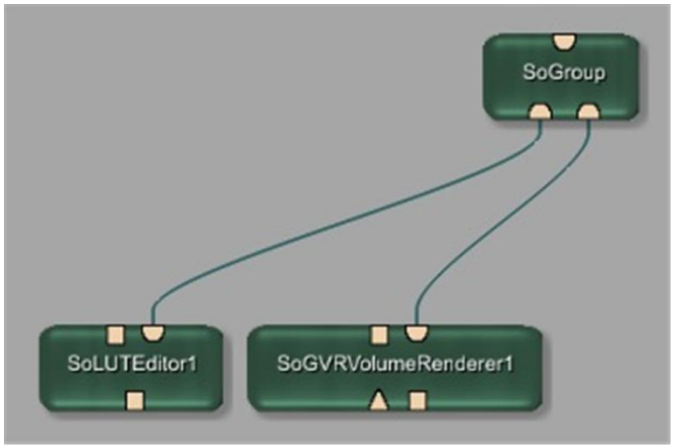

Abb. 4.6: Verbindung des SoLUTEditors und des SoGVRVolumeRenderer mit SoGroup (Knogler, Panzenböck, 2015)

Die **Abb. 4.6** zeigt ein Beispiel für die Verbindungen des SoLUTEditor und des Renderers.

Gruppierung -> Panel SoGroup

Um die Verbindung zwischen dem Editor und dem Renderer herzustellen, wird das Modul SoGroup verwendet.

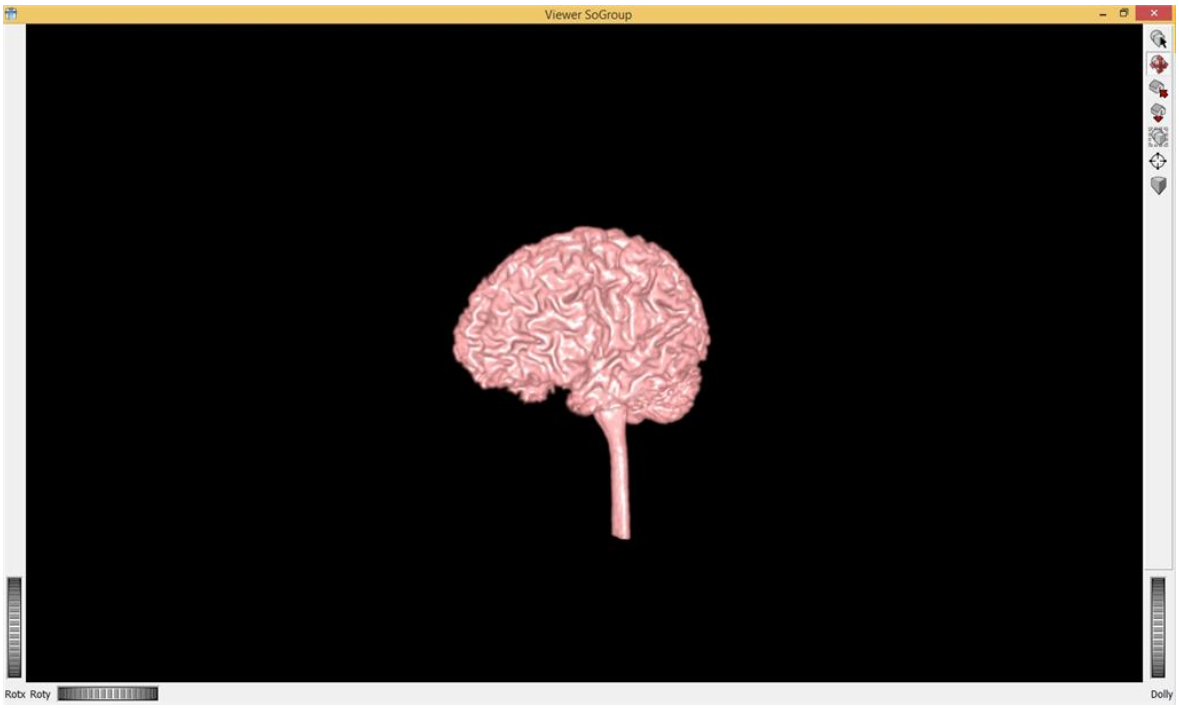

Abb. 4.7: Darstellung des Hirnmarks (Knogler, Panzenböck, 2015)

Das Ergebnis der Segmentierung des Hirnmarks inklusive Filterung und Volume Rendering wird über das Modul „SoGroup“ (siehe **Abb. 4.7**) dargestellt.

4.4. Import neuer Anleitungen

Eines der primären Ziele von MLA ist es, einfach und schnell, neue Anleitungen zur Anwendung hinzuzufügen. Durch diese Erweiterbarkeit gewinnt die Applikation an Nutzen und kann länger genutzt werden.

Um neue Dokumente zu importieren muss, zuerst auf den "Einstellungen" Button geklickt werden (Punkt 10 auf **Abb. 4.3** Auswahl eines Beispiels).

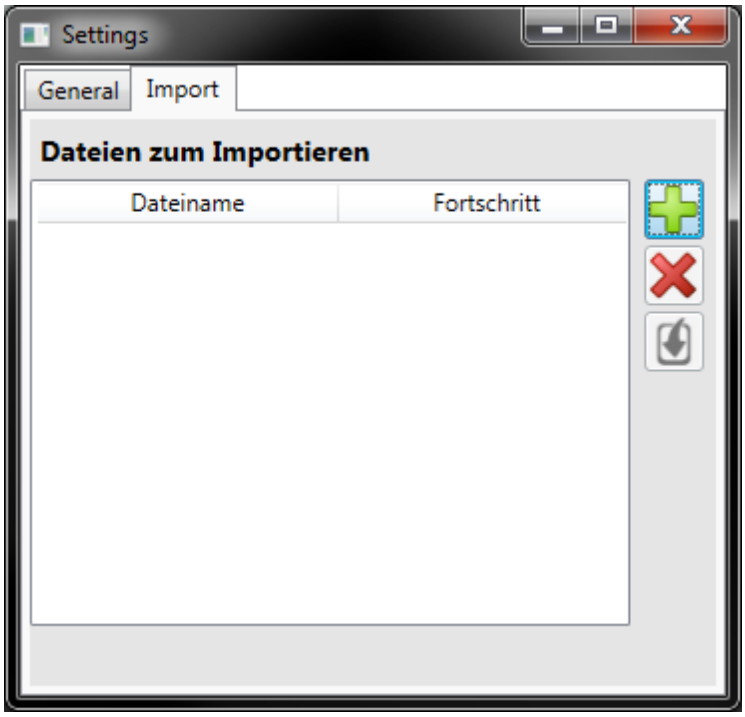

Abb. 4.8 Einstellungs-Fenster

Wie auf **Abb. 4.8** Einstellungs-Fenster zu sehen ist, muss auf den Reiter "Import" geklickt werden. Um nun Dokumente hinzuzufügen, muss auf den Button mit dem Plus geklickt werden. Ein Datei Dialog öffnet sich.

Der Datei Dialog lässt lediglich ".docx" Dateien, also Microsoft Word Dokumente im XML Format zu. Hier können beliebig viele Dokumente ausgewählt werden.

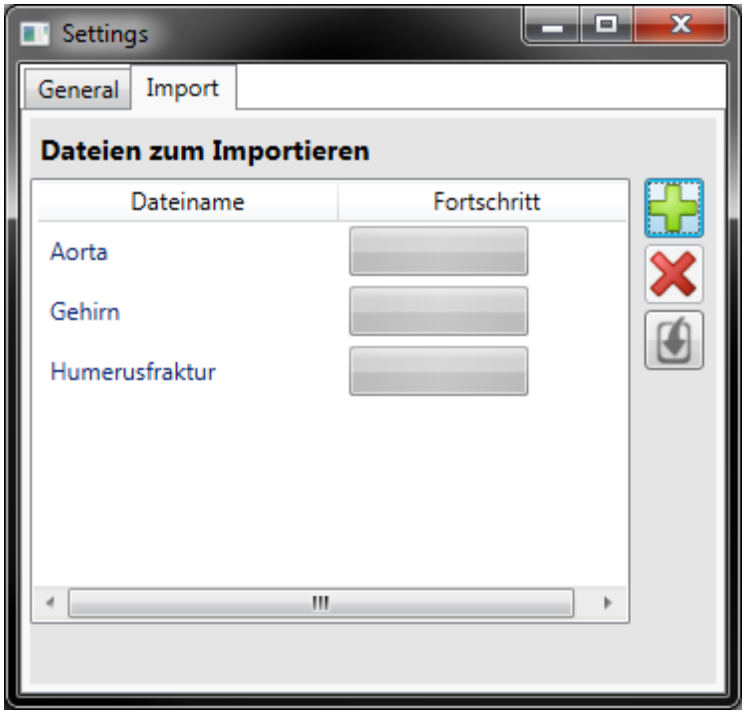

Abb. 4.9 Auswahl der Dokumente

Sollte nun der Bedarf bestehen, Dokumente wieder aus der Auswahl zu löschen, muss das gewünschte Dokument selektiert werden und dann auf den Button mit dem "X" geklickt werden (siehe **Abb. 4.9** Auswahl der Dokumente)

Um den Import endgültig durchzuführen, muss man auf den Button mit dem Pfeil nach unten klicken und warten, bis alle Fortschrittbalken vollständig ausgefüllt sind (siehe **Abb. 4.10** Import beendet).

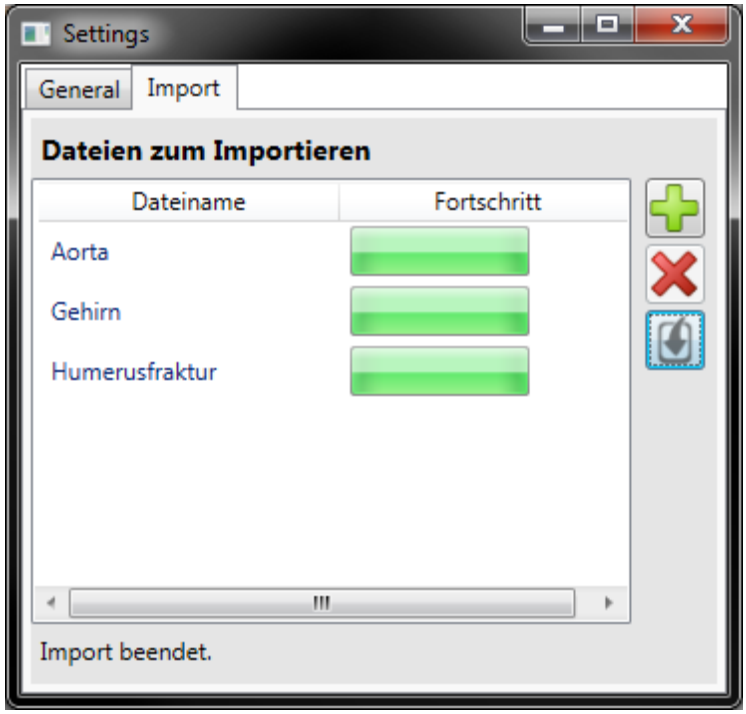

Abb. 4.10 Import beendet

Der Import Dialog kann danach geschlossen werden und die Dokumente stehen im Hauptfenster zur Verfügung.

5. Programmstruktur

5.1. Architektur

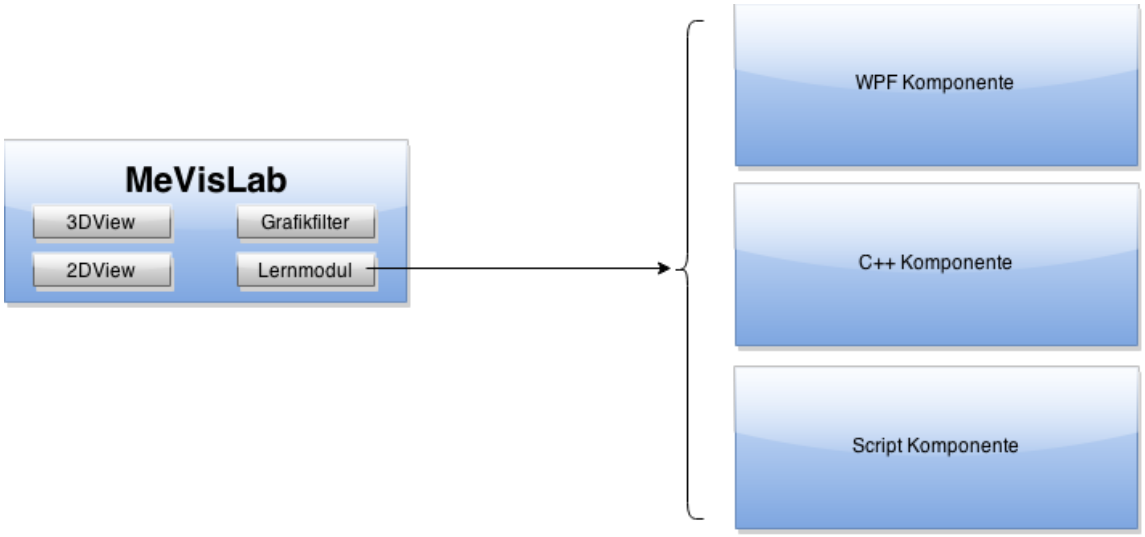

Abb. 5.1 Architektur

In **Abb. 5.1** Architektur ist die Systemarchitektur skizziert. Man kann erkennen, dass das Programm MeVisLab aus verschiedenen Modulen besteht. Eines davon wurde im Rahmen dieser Diplomarbeit entwickelt, um ein Lernprogramm aufzurufen. Dieses Modul besteht aus einem C++ Programm, welches die Schnittstelle zwischen Logik und Benutzeroberfläche regelt – in diesem Fall ruft es die WPF Anwendung auf. Für die Darstellung der Oberfläche des Moduls ist die Script Komponente zuständig. Die WPF Anwendung ist der wesentliche Teil dieser Diplomarbeit, denn sie deckt die wesentlichen Instruktionen für das Erlernen von Bildverarbeitungsmechanismen ab.

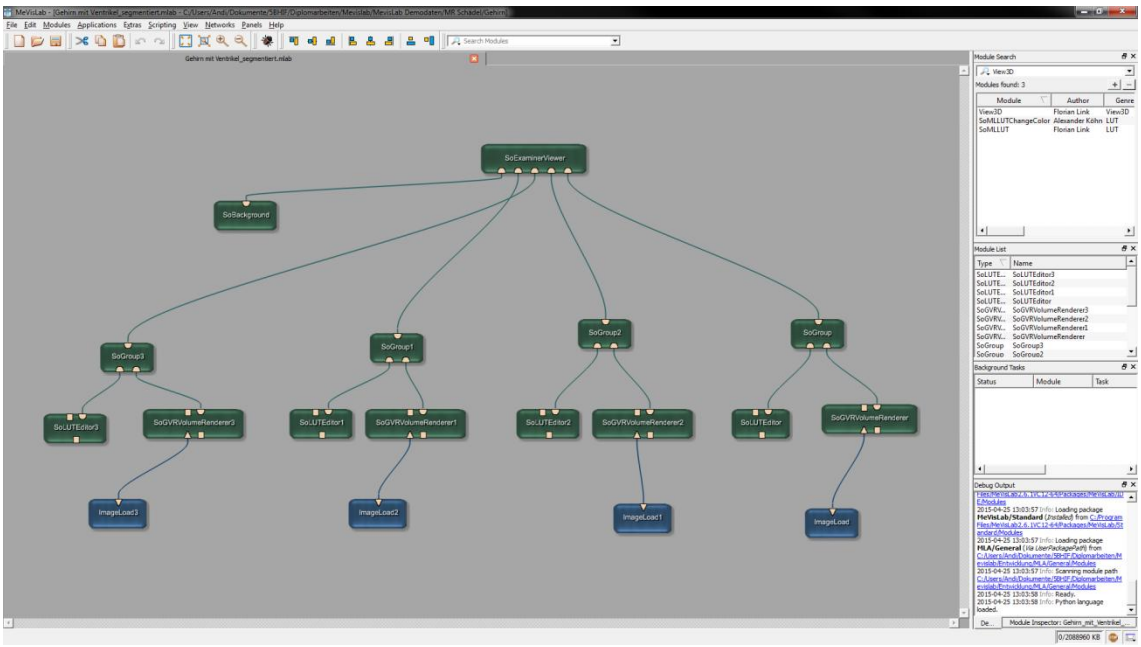

Abb. 5.2 Module von MeVisLab

Abb. 5.2 Module von MeVisLab stellt dar, wie dieses Modulsystem in MeVisLab in der Praxis aussieht. Jeder färbige Kasten im Bild stellt ein Modul dar.

5.2. Klassendiagramm WPF

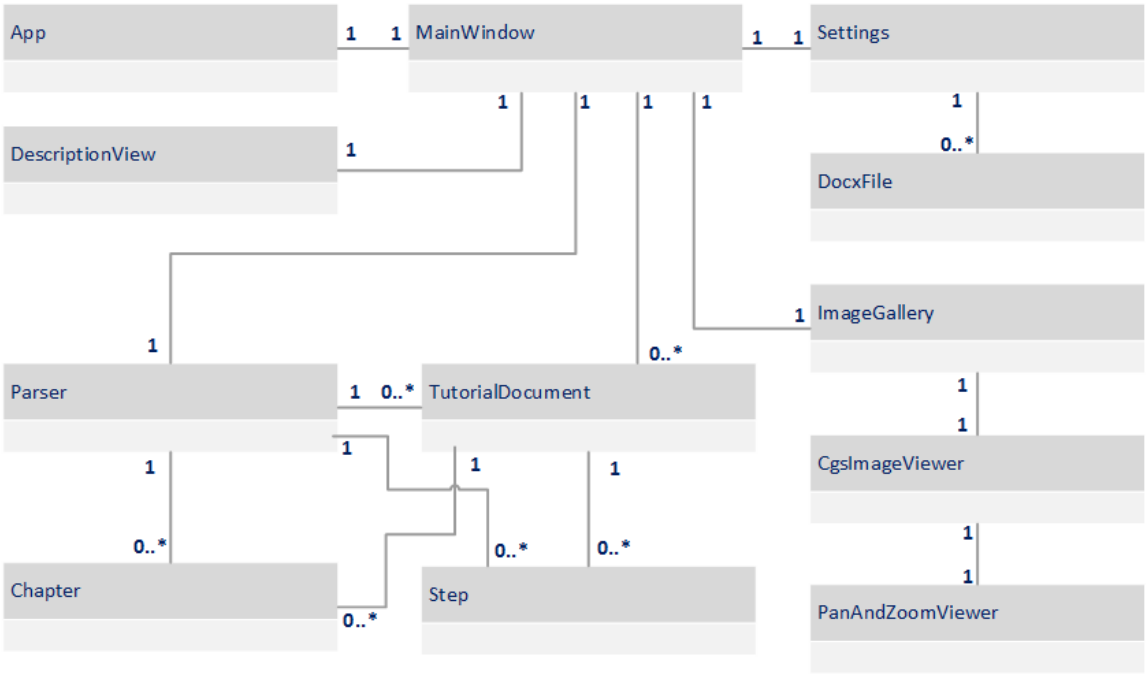

Abb. 5.3 Klassendiagramm des C# Codes

In diesem Klassendiagramm (**Abb. 5.3**) kann man die einzelnen Klassen und deren Attribute sowie die Beziehungen untereinander finden.

Die Klasse „*App*“ dient nur zum Initialisieren des WPF Programmes und wird automatisch von Visual Studio angelegt. Das „*MainWindow*“ ist die zentrale Klasse des WPF Programmes. Sie verwendet ein Objekt der Klasse „Parser“ zum Einlesen von Anleitungen und verschiedene Objekte der Klasse „*TutorialDocument*“ zum Darstellen der vorhandenen Anleitungen in der Benutzeroberfläche. Die Klasse „*Parser*“ braucht die Klassen „*TutorialDocument*“, „*Step*“ und „*Chapter*“ zum Speichern des Inhaltes der Anleitungen. Rechts oben im Klassendiagramm ist außerdem noch die Klasse „*Settings*“ zu sehen, die es ermöglicht, neue Anleitungen zur Laufzeit hinzuzufügen. Zur temporären Speicherung wird die Klasse „*DocxFile*“ herangenommen. Damit Bilder und deren Buttons zum Navigieren dargestellt werden können, werden die Klassen „*ImageGallery*“, „*CgsImageViewer*“ und „*PanAndZoomViewer*“ verwendet.

Im Folgenden werden nun die Klassen erklärt.

5.2.1. App

Abb. 5.4 App

Die bei der Erstellung eines WPF Projekts vorgegebene Klasse "*App*" (siehe **Abb. 5.4** App) beinhaltet die zum Start des Programms notwendige "*Main*"-Methode. Hier erfolgt der Aufruf der Klasse "*MainWindow*", in der die Initialisierung der Oberfläche stattfindet.

5.2.2. Parser

Die Klasse „Parser“ (siehe **Abb. 5.5** Parser) ist einer der Hauptbestandteile der WPF Anwendung. Hier werden „*.docx*“ Dateien in HTML Dokumente umgewandelt. Dies geschieht mithilfe der Microsoft Office Word Bibliothek „Microsoft.Office.Interop.Word“, was aber voraussetzt, dass das eben genannte Produkt auch am selben Rechner installiert ist. Anschließend werden die HTML Dokumente unter Verwendung von „*Html Agility Pack*“ ausgelesen und in den Zwischenspeicher geladen. Der "Parser" unterscheidet zwischen Dokumententitel, Kapitelnamen und -beschreibung, Schritten sowie deren Überschrift und Abbildungen inklusive derer Beschriftungen.

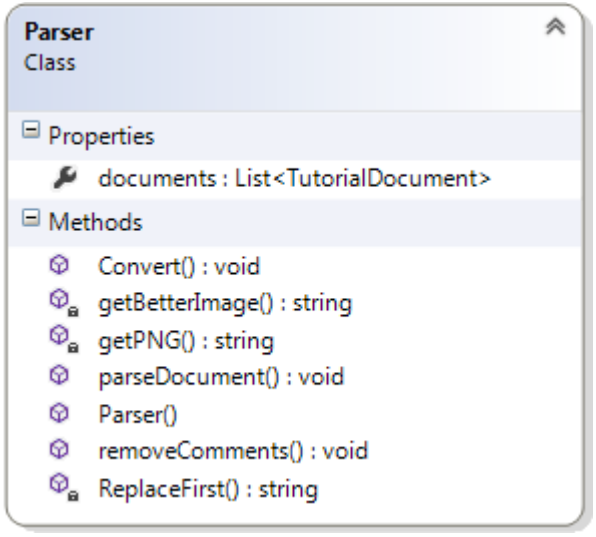

Abb. 5.5 Parser

5.2.3. TutorialDocument

Der vorher beschriebene *Parser* verwendet zum Zwischenspeichern Objekte der Klasse „*TutorialDocument*“ (siehe **Abb. 5.6** TutorialDocument). Ein „*TutorialDocument*“ stellt eine Step-by-Step Word-Anleitung dar. In diesem "*TutorialDocument*" werden die jeweiligen Kapitel "*Chapters*" und Schritte "*Steps*" zwischengespeichert. Darüber hinaus wird hier der Name des Dokumentes eingefügt.

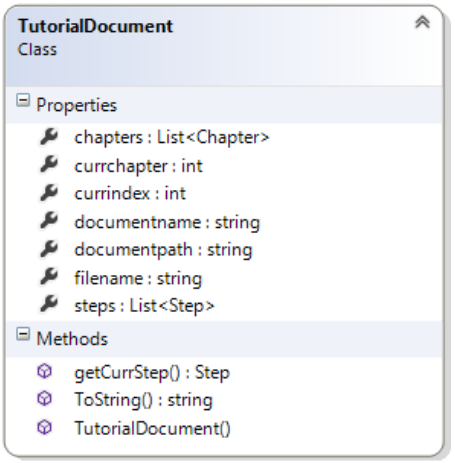

Abb. 5.6 TutorialDocument

5.2.4. Chapter

Der Parser extrahiert die einzelnen Kapitel aus der "*docx*"-Datei und erstellt daraus ein Objekt der Klasse "*Chapter*" (siehe **Abb. 5.7**), welches sämtliche Attribute eines Kapitels beinhaltet. Dieses Objekt wird anschließend in ein Objekt der Klasse "*TutorialDocument*" eingefügt.

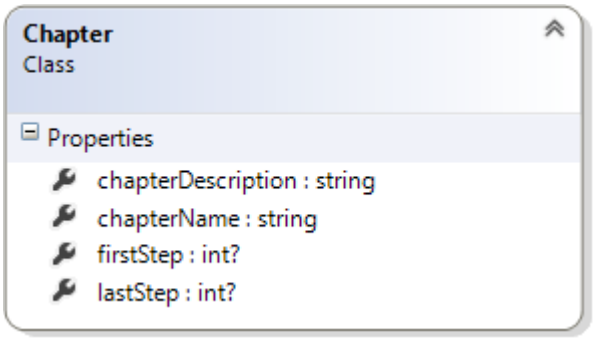

Abb. 5.7 Chapter

5.2.5. Step

Die Klasse "*Step*" (siehe **Abb. 5.8** Step) stellt einen Schritt in der "*docx*"-Datei dar und definiert sämtliche Attribute eines "*Step*". Dieses Objekt wird vom Parser in ein Objekt der Klasse "*TutorialDocument*" eingefügt.

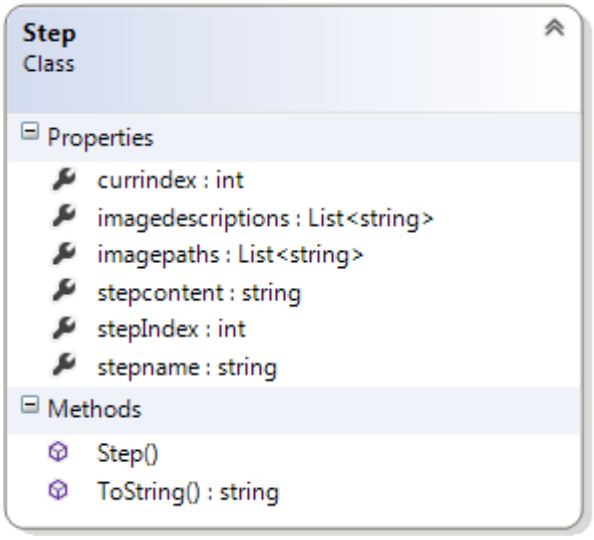

Abb. 5.8 Step

5.2.6. MainWindow

Das „*MainWindow*“ (siehe **Abb. 5.9** MainWindow) bezeichnet das Fenster, welches den Kern der WPF Anwendung bildet. Es besteht aus „*XAML*“-Code und „*Code-behind*“ – also dem C# Code. Im „XAML“-Code werden alle Buttons, Textboxen und sonstige Oberflächenelemente festgelegt und im „Code-behind“ verbirgt sich die Logik, also was z.B. passieren soll, wenn ein Button gedrückt wird. Beim ursprünglichen Aufruf von "*MainWindow*" wird ein Objekt der Klasse "*Parser*" erzeugt und somit die HTML Dokumente eingelesen.

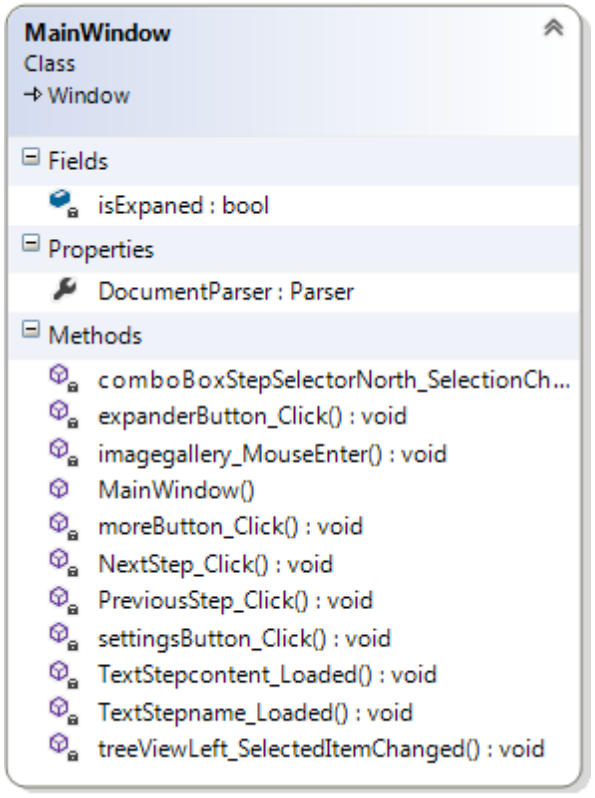

Abb. 5.9 MainWindow

5.2.7. DescriptionView

Da die Anleitungen im "*docx*" Format vorgeben, dass ein Kapitel eine Beschreibung haben kann, stellt die Klasse "*DescriptionView*" (siehe **Abb. 5.10** DescriptionView) diese Funktionalität zur Verfügung. Die vom "Parser" ausgelesenen Beschreibungen können individuell pro selektiertem Kapitel in einem separaten Fenster angezeigt werden.

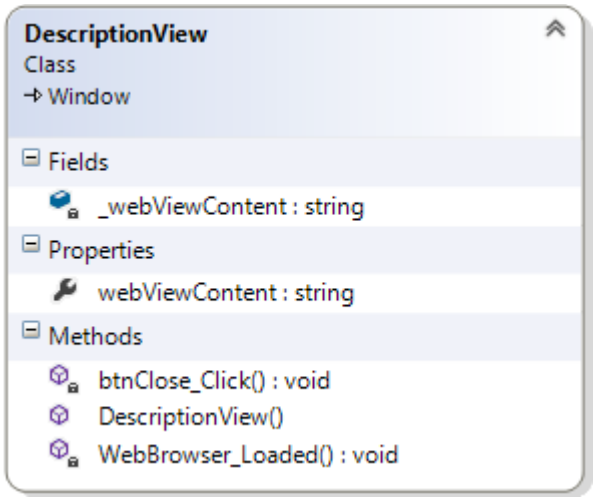

Abb. 5.10 DescriptionView

5.2.8. Settings

Die Klasse "Settings" (siehe **Abb. 5.11** Settings) definiert ein Fenster, welches es ermöglicht, weitere "*docx*"-Dokumente einzulesen. Wie die Klasse "*MainWindow*" besteht die Klasse "*Settings*" aus "*XAML*"-Code und "*Code-behind*". Der Zugriff auf das "*Parser*"-Objekt der Klasse "*MainWindow*" macht es möglich, die weiteren "*docx*"-Dateien umzuwandeln und einzulesen.

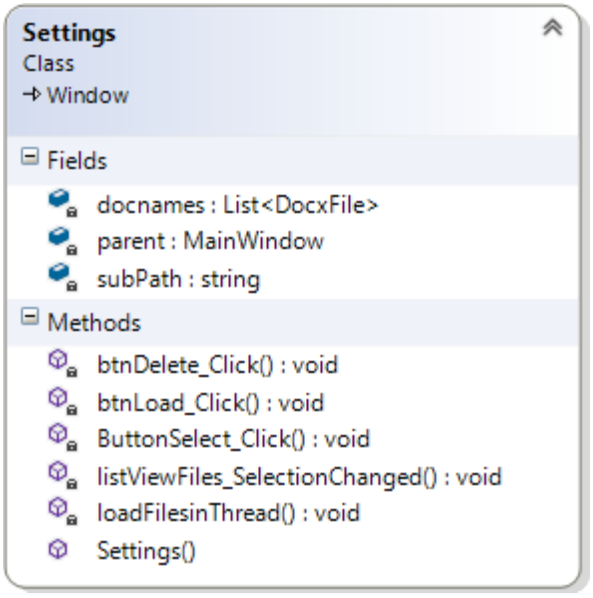

Abb. 5.11 Settings

5.2.9. DocxFile

Die Klasse "*DocxFile*" (siehe **Abb. 5.12** DocxFile) stellt sämtliche Attribute, die zum Auslesen eines Microsoft Word-Dokuments benötigt werden, vor (Der Pfad zum Dokument, , der Dateiname und der Fortschritt des Einlesens). Sie dient nicht nur zum Auffinden von Dokumenten, sondern gibt dem Benutzer auch Auskunft darüber, wie weit der Fortschritt des Imports ist.

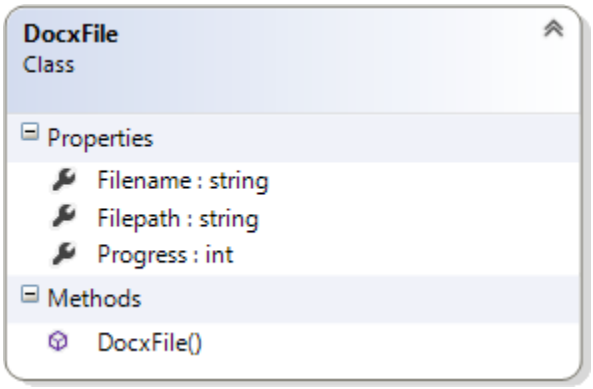

Abb. 5.12 DocxFile

5.2.10. CgsImageViewer, PanAndZoomViewer und ImageGallery

Diese drei Komponenten (siehe **Abb. 5.13** Bildanzeige) ermöglichen das ordnungsgemäße Anzeigen von Bildern im oben erwähnten „*MainWindow*“. „*ImageGallery*“ hat ein *UserControl* (ein individuell definier- und parametrierbares „*Control*“), nämlich konkret den „*CgsImageViewer*“. Dieser benutzt ein Objekt der Klasse „*PanAndZoomViewer*“, welcher – wie der Name bereits verrät – das Zoomen und Umherziehen eines Bildes erlaubt.

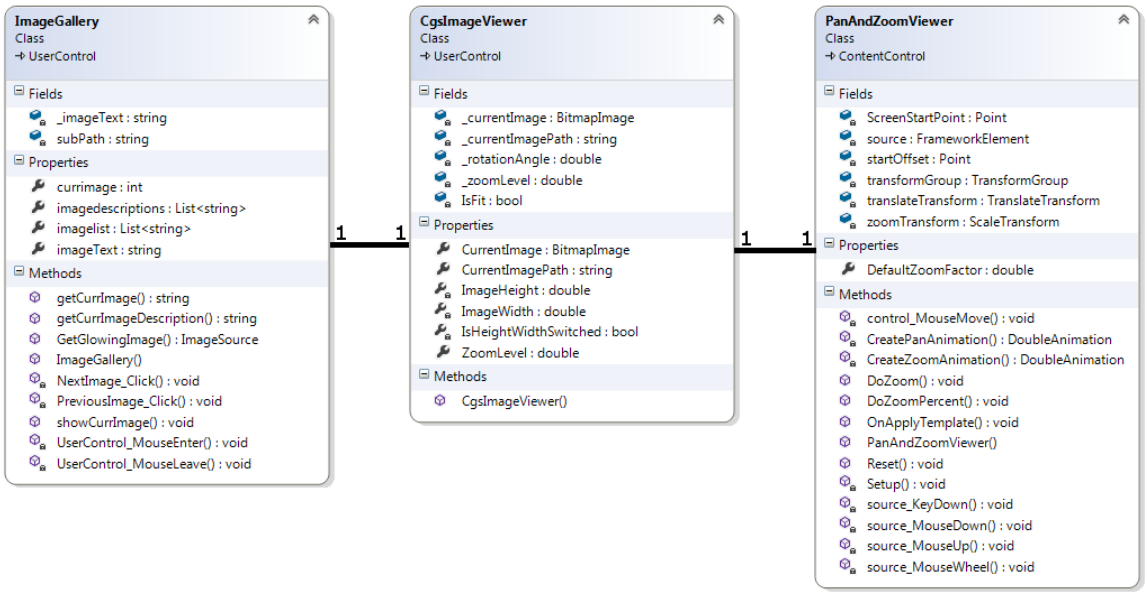

Abb. 5.13 Bildanzeige

5.3. Klassendiagramm C++

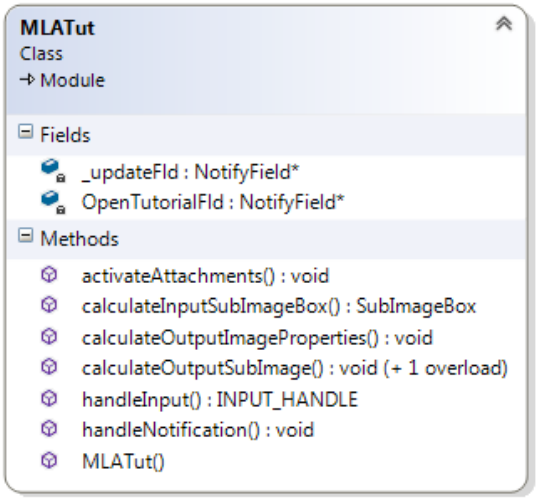

Abb. 5.14 Klassendiagramm C++

Das Klassendiagramm (siehe **Abb. 5.14** Klassendiagramm C++) des C++ Projektes besteht im Grunde aus einer Klasse, die die Eingabe- und Ausgabeoperationen des MeVisLab Moduls erfüllt. Speziell in der Methode „handleNotification“ wird hier das WPF Programm aufgerufen, das den wesentlichen Teil des Gesamtprojekts bildet.

6. Implementierung

6.1. MeVisLab Komponente

In diesem Kapitel wird erklärt, was zum Erzeugen eines MeVisLab Moduls notwendig ist.

6.1.1. Paket anlegen

Für selbst entwickelte Module sollte immer ein eigenes Paket angelegt werden. Um ein solches Paket (File -> Run Project Wizard -> new Package) zu erzeugen, ist es notwendig den „Project Wizard“ zu starten, welchen man in MeVisLab im Reiter „File“ findet. Von dieser Maske aus kann ein neues Paket bzw. auch ein neues Modul erstellt werden.

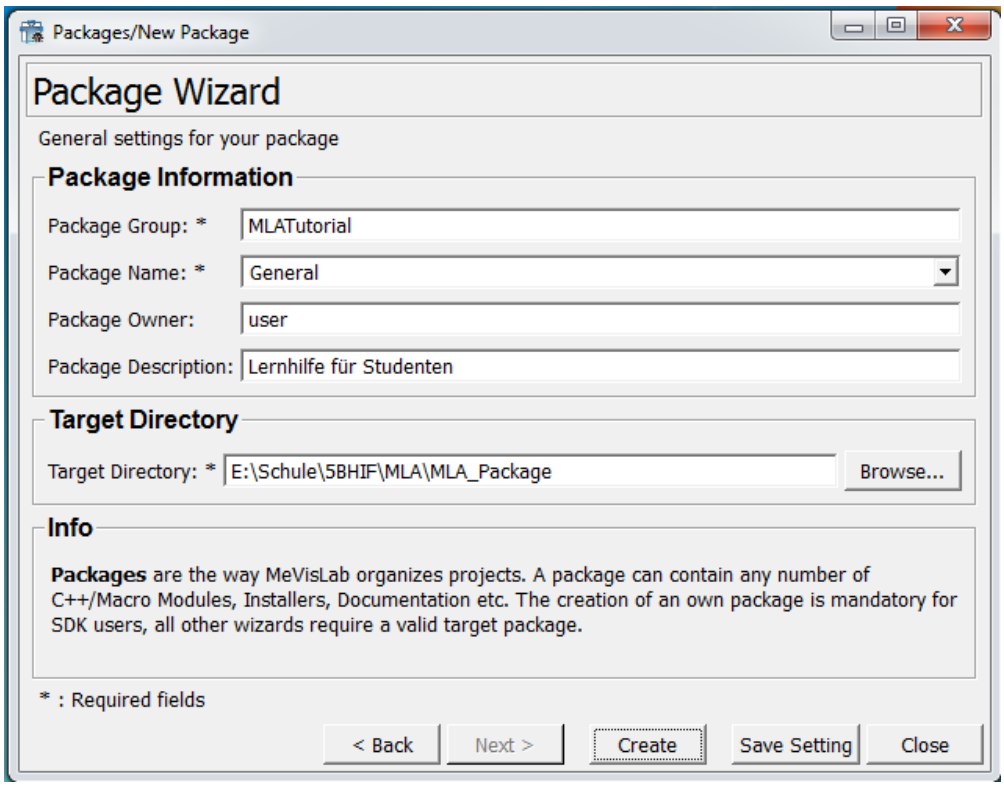

Abb. 6.1 Package Wizard

In **Abb. 6.1** Package Wizard ist die Oberfläche für die Installation eines neuen Paketes zu sehen. Wurde das Paket erfolgreich zusammengestellt, so stellt man fest, dass im angegebenen Pfad eine Reihe von Ordnern und Dateien erzeugt wurde. (siehe **Abb. 6.2** Ordnerstruktur des Paketes) Alle neu erstellten Module werden in dieses Paket integriert [2003-2012].

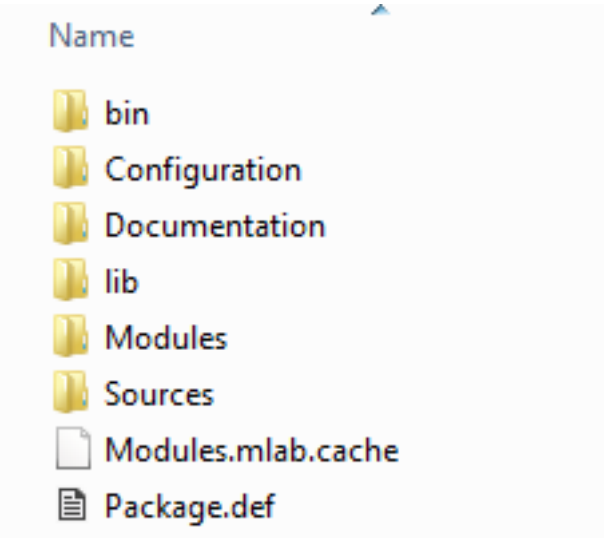

Abb. 6.2 Ordnerstruktur des Paketes

So werden beispielsweise im Ordner „Modules“ jegliche Scriptkomponenten, die für das Aussehen des Modules zuständig sind, gespeichert. Die Funktionalität, welche meistens in Form eines C++ Projektes erzeugt wird, ist im Ordner „Sources“ wiederzufinden.

6.1.2. ML Modul anlegen

Für das Anlegen eines Moduls muss zuerst wieder der „Project Wizard“ von MeVisLab gestartet werden. Anschließend wird das ML Modul ausgewählt und eine Oberfläche zum Erzeugen eines Moduls erscheint. Wichtig hierbei ist, dass das im vorigen Kapitel angefertigte Paket ausgewählt wird.

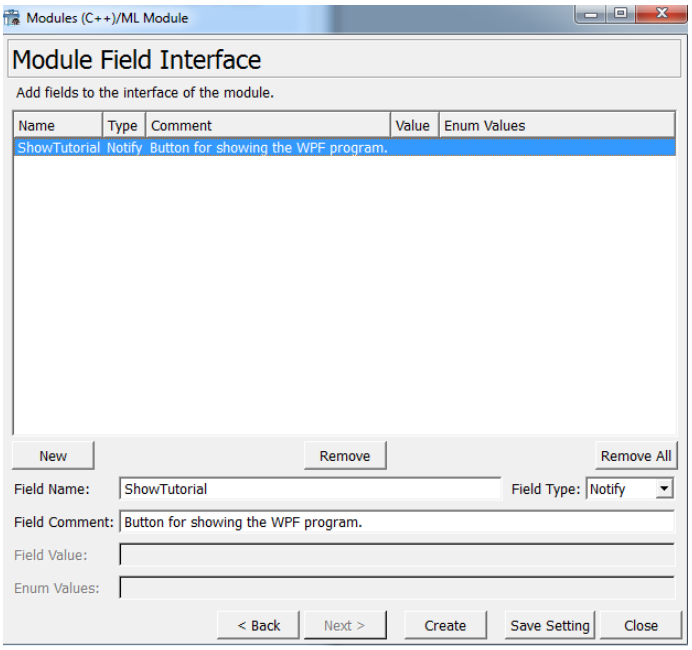

Abb. 6.3 Definition der Oberflächenelemente

Das einzige Oberflächenelement, das vorausgesetzt wird, ist ein Feld des Typs „Notify“, das als Button fungiert. (siehe **Abb. 6.3** Definition der Oberflächenelemente**)** Dieses Feld ist dafür zuständig, den dahinterliegenden C++ Code aufzurufen, welcher wiederum für das Öffnen des WPF Programmes erforderlich ist. Beim Erzeugen des Moduls öffnen sich zwei Windows-Explorer Fenster, die jeweils das automatisch zusammengestellte C++ Projekt bzw. die Script Dateien für die Oberfäche anzeigen.

6.1.3. Entstandene Script Dateien

Wie bereits kurz erwähnt, wurden zwei **MDL** Script Dateien von MeVisLab angefertigt.

- _MLmlatutorial.def
- MLATut.script

Das Script mit der Endung „*.def*“ wird von MeVisLab dafür verwendet, das Modul in die lokale Moduldatenbank zu integrieren. Enthalten sind Attribute wie Genre, Autor, Kommentar, sowie der Name der zuständigen DLL.

Das andere Script, welches mit „*.script*“ endet, beschäftigt sich mit der Oberfläche des Moduls und deren Elemente.

```
1  Window {
2    Vertical {
3      margin = 3
4      Field ShowTutorial {
5          title = "Open Tutorial"
6      }
7    }
8  }
```

Codebeispiel 6.1 Inhalt des Scripts „MLATut.script“

Wie in Zeile 1 der **Codebeispiel 6.1** Inhalt des Scripts „MLATut.script“ zu sehen ist, wird hier das gesamte Modulfenster definiert, wobei in Zeile 2 anschließend eine vertikale Anordnung angedeutet wird. Zeile 3 verhindert, dass der Inhalt am oberen Rand des Fenster andockt. Das Feld „*ShowTutorial*“, wie in Zeile 4 zu sehen ist, definiert das zuvor in Kapitel **6.1.2** gesetzte Feld des Typs „*Notify*“.

6.2. C++ Komponente

Beim Erzeugen des ML Moduls konstruiert das Programm MeVisLab ein C++ Projekt, das den wesentlichen Source-Code enthält. Zu finden sind Methoden wie „`calculateInputSubImageBox`“, „`calculateOutputSubImage`“, „`handleInput`“ oder „`handleNotification`“, welche bereits eine gewisse Logik durchführen und auch vorausgesetzt sind, da von der Klasse „`Module`“ abgeleitet wird. Natürlich können diese umprogrammiert werden bzw. können auch zusätzliche Funktionalitäten hinzugefügt werden.

```
1  STARTUPINFO si = { sizeof(STARTUPINFO) };
2  si.cb = sizeof(si);
3  si.dwFlags = STARTF_USESHOWWINDOW;
4  si.wShowWindow = SW_HIDE;
5  PROCESS_INFORMATION pi = { "mlaID" };
6  wchar_t* file = L"Packages/MLA/MLAWindow.exe";
7  if (!IsProcessRunning(GetWC("MLAWindow.exe")))
8  {
9    CreateProcess(file, NULL, NULL, NULL, FALSE,
10   CREATE_NO_WINDOW, NULL, NULL, &si, &pi);
11 }
```

Codebeispiel 6.2 Aufruf des WPF Programms in C++

Wie in **Codebeispiel 6.2** Aufruf des WPF Programms in C++, Zeile 1 und 2, zu sehen ist, wird eine Variable der Struktur „STARTUPINFO“ deklariert, initialisiert sowie dessen Größe festgelegt. In Zeile 3 wird ein Flag gesetzt, damit die Zuweisung in Zeile 4 überhaupt in Kraft treten kann, denn diese setzt die Variable „`wShowWindow`“ auf den Wert „`SW_HIDE`“, um zu verhindern, dass ein CMD-Fenster beim Starten des Prozesses aufpoppt [Microsoft MSDN].

In Zeile 5 wird eine neue Variable der Struktur „*PROCESS_INFORMATION*“ angelegt, dessen Aufgabe es ist, Informationen über und für den Prozess an den Prozess weiterzugeben. In diesem Fall wird nur die ID des Prozesses definiert [Microsoft MSDN].

Da das doppelte Starten des WPF Programmes unerwünscht ist, wird in Zeile 7 geprüft, ob der Prozess zuvor gestartet wurde und noch läuft. Wenn dies nicht der Fall ist, kann das Lernprogramm mittels der Methode "CreateProcess" in Zeile 9 und 10 gestartet werden.

Für das Arbeiten mit Prozessen in C++ muss die Header Datei „Windows.h“ inkludiert werden.

6.3. C# Komponente

Nach dem Aufruf von "MLAWindow.exe" in C++ beginnt der C# Teil.

6.3.1. Start der Applikation

Die Main()-Methode befindet sich in der Klasse "App", von hier aus beginnt der Aufruf der Applikation.

```
1  public void InitializeComponent() {
2    #line 4 "..\..\App.xaml"
3    this.StartupUri = new System.Uri(
4    "MainWindow.xaml", System.UriKind.Relative);
5    #line default
6    #line hidden
7  }
```

Codebeispiel 6.3 Start der Applikation

In der Main()-Methode wird "InitializeComponent" (siehe **Codebeispiel 6.3** Start der Applikation) aufgerufen, welche, wie in **Zeile 4** ersichtlich, die Klasse "MainWindow" aufruft.

6.3.2. Initialisierung

In der Klasse "MainWindow" angelangt, werden zuerst alle bestehenden HTML Dokumente im Unterordner "Documents" importiert.

```
1  DirectoryInfo d = new DirectoryInfo(
2    @System.Environment.CurrentDirectory + "/" + subPath);
3  FileInfo[] Files = d.GetFiles("*.html");
4  foreach (FileInfo file in Files)
5  {
6    DocumentParser.parseDocument(
7    @System.Environment.CurrentDirectory + "/" +
8    subPath + "\\"+file.Name);
9  }
10 DataContext = DocumentParser.documents;
```

Codebeispiel 6.4 Import der HTML Dateien

In **Zeile 1 und 2** der **Codebeispiel 6.4** Import der HTML Dateien wird der Unterordner "Documents" "geöffnet" und in **Zeile 4** durchlaufen. In **Zeile 6,7 und 8** übernimmt der Parser den Import der einzelnen Dateien.

6.3.1. Einlesen von Word-Dokumenten

Die Klasse „*Parser*“ beschäftigt sich mit dem Konvertieren von „*.docx*“-Dateien in „*.html*“ Dateien sowie dem Einlesen des Inhaltes.

Für das Konvertieren wird die C# Bibliothek von Microsoft Office Word verwendet. Mithilfe der Bibliothek wird das Programm Office Word im Hintergrund gestartet und anschließend wird eine Word-Anleitung hineingeladen. Im konkreten Fall wird hierbei die statische Methode „*Application()*“ aufgerufen, welche ein neues Objekt von Word zurückgibt. Natürlich ist es nicht sehr interessant für den Benutzer, wie das Programm dieser Diplomarbeit im Hintergrund arbeitet, darum kann die Variable „Visible“ des neu erstellten Objektes von Word auf „*false*“ gesetzt werden, damit Office Word nicht sichtbar für den Benutzer ist.

Da nun ein Prozess für Office Word gestartet wurde, kann die tatsächliche Anleitung mit Zugriff auf die Collection „*Documents*“ und dessen Methode „*Open()*“ geöffnet werden. Daraufhin wird das Dokument mit der Methode „SaveAs()“, wobei als Parameter der Dateityp („*.html*“) übergeben wird, gespeichert.

Für das eigentliche Einlesen des Inhaltes wird die Bibliothek „HtmlAgilityPack“ verwendet. Das Hereinladen der „*.html*“ - Anleitungen erfolgt nach dem Schema, das in Kapitel **2.4.5** erklärt wird.

```
1  HtmlNode documentnode = null;
2  foreach (HtmlNode node in
3            doc.DocumentNode.SelectNodes("//h1"))
4  {
5          docbean.documentname =
6          System.Net.WebUtility.HtmlDecode(node.InnerText);
7          documentnode = node;
8  }
9  Boolean isAlreadyAChapter = false;
10 while (!isAlreadyAChapter)
11 {
12    documentnode =documentnode.NextSibling;
13    if(documentnode != null &&
14       documentnode.Name.Equals("h2"))
15    {
16        isAlreadyAChapter = true;
17    }
18    else
19    {
20       docbean.documentdescription +=
21         ReplaceFirst(documentnode.InnerText,
22                      " ", "")
23         .Replace("\n","").Trim();
24     }
25 }
```

Codebeispiel 6.5 Einlesen des Dokumententitels

Das Einlesen beginnt mit dem Dokumententitel, der im HTML-Code in ein „<h1>“ Tag eingebettet ist. Im **Codebeispiel 6.6,** Zeile 2 und 3 ist zu sehen, dass alle Dokumententitel (also alle „<h1> Tags) selektiert und in Zeile 5-6 anschließend in das Objekt „*docbean*“ der Klasse „*TutorialDocument*“ geschrieben werden. Wenn der Titel erfolgreich eingetragen wurde, sucht der Parser nun nach der Beschreibung dazu. Nach Einlesen des Dokumententitels wird nach der Beschreibung dazu gesucht, die unmittelbar danach auftauchen sollte (siehe Zeile 9-25). In Zeile 13-17 ist die Abfrage zu sehen, die prüft, welcher Typ der aktuelle Dokumentenknoten „*documentnode*“ ist. Falls es ein „<h2>“ Tag ist, wird angenommen, dass keine Beschreibung vorhanden ist, da ein „<h2>“ Tag bereits als Kapitelname definiert wurde. Eine Ähnliche Vorgehensweise wird auch bei den Kapiteln und Schritten angewandt.

```
1  if (helpnode.SelectNodes(".//img")
2  {
3        String x =
4        getBetterImage
5        (
6           System.Net.WebUtility.HtmlDecode(
7               helpnode.SelectNodes(".//img")[0]
8               .Attributes["src"].Value)
9        );
10       stepX.imagepaths.Add(x);
11       stepX.imagedescriptions.Add(
12          helpnode.NextSibling.NextSibling
13          .InnerText.Trim());
14 }
```

Codebeispiel 6.6 Einlesen der Bilder

Das Einlesen von Bildern in den Anleitungen erfolgt wie in **Codebeispiel 6.6.**

Wenn der aktuelle Dokumentenknoten „*helpnode*“ vom Typ „*<img>*“ ist und nicht leer ist, so wird der Pfad des Bildes, wie in Zeile 3 - 9 zu sehen ist, in eine String-Variable gespeichert. Die Methode „*System.Net.WebUtility.HtmlDecode*“ dient dazu, HTLM-Codierte Zeichen (wie z.B. ein Pfeil →) zu erkennen und richtig abzuspeichern. In Zeile 8 ist zu sehen wie lediglich der Wert des Attributes „*src*“ ausgelesen wird und keine weiteren unnötigen Informationen. Natürlich wird das Bild nicht während des Einlesens eines Dokumentes in den Arbeitsspeicher geladen, da, bei unzähligen Bildern, die Performance stark darunter leidet. In Zeile 10-13 ist schließlich noch zu sehen, wie der Bildpfad abgespeichert wird. „*stepX*“ ist in diesem Fall ein Objekt der Klasse „*Step*“ und speichert Informationen über den aktuellen Schritt.

6.3.2. Import von neuen Dokumenten

Der für den Import von neuen Beispielen wichtige Code befindet sich in der Klasse "Settings". In der Methode "ButtonSelect_Click" befindet sich der Dateidialog, der so konfiguriert ist, dass er nur ".docx" Dokumente entgegennimmt.

```
1  Microsoft.Win32.OpenFileDialog dlg =
2  new Microsoft.Win32.OpenFileDialog();
3  dlg.FileName = "Document";
4  dlg.DefaultExt = ".docx";
5  dlg.Filter = "Word documents (.docx)|*.docx";
6  dlg.Multiselect = true;
7  Nullable<bool> result = dlg.ShowDialog();
8  if (result == true)
9  {
```

```
10   for (int i =0;i<dlg.SafeFileNames.Count();i++)
11   {
12   docnames.Add(new DocxFile(dlg.SafeFileNames[i].Replace
13   (".docx", ""),0,dlg.FileNames[i]));
14   }
15   listViewFiles.ItemsSource = docnames;
16   listViewFiles.Items.Refresh();
17   btnImport.IsEnabled = true;
18 }
```

Codebeispiel 6.7 Import von Dokumenten

In **Codebeispiel 6.7** Import von Dokumenten, **Zeile 1-6** wird ein OpenFileDialog initialisiert und so konfiguriert, dass mehrere ".docx" Dateien entgegengenommen werden können.

Aus den Zeilen 10-14 ist zu entnehmen, dass das Objekt der Klasse OpenFileDialog durchlaufen wird und alle selektierten Dokumente ausgelesen und ohne Dateiendung in eine eigene Liste gespeichert werden.

Die Zeilen 15-16 sorgen dafür, dass die Dokumente im Fenster angezeigt werden (siehe **Abb. 4.9** Auswahl der Dokumente).

6.3.3. Selektieren der richtigen Bilder

Bei der Umwandlung von Word Dokumenten in HTML Dokumente werden Bilder in komprimierter Version und in Originalgröße generiert. Dies geschieht jedoch nicht mit allen Bildern. Manchmal wird nur eines generiert, manchmal sogar drei und dann jeweils in unterschiedlichen Bildformaten. Daher war es notwendig, einen verlässlichen Algorithmus zu finden, welche die höherwertigen Bilder herausfiltert.

Beim Parser wird beim Fund eines Bildes die Methode "*getBetterImage()*" aufgerufen.

Hier wird zuerst der Dateiname von der Dateiendung getrennt. Danach passiert folgendes:

```
1  if(newimage == getPNG(newimage))
2  {
3    if(File.Exists("Documents/" + newimage) &&
4    isSameImage(newimage, image))
5    {
6      return newimage;
7    }
8    else
9    {
10     return image;
```

```
11   }
12 }
13 else
14 {
15   return (getPNG(newimage));
16 }
```

Codebeispiel 6.8 Rückgabe des besseren Bildes

Wie in **Codebeispiel 6.8** Rückgabe des besseren Bildes Zeile 4 zu sehen ist, wird beim gefundenen Bild verglichen, ob das vom Algorithmus als bessere Bild interpretierte Bild tatsächlich dasselbe ist, nur in besserer Qualität. Dazu wird der Code wie in **Codebeispiel 3.3** verwendet.

Um Bilder überhaupt vergleichen zu können, müssen sie exakt gleich groß sein, weil Pixel für Pixel verglichen wird.

```
1  Bitmap bitmap1 =
2  new Bitmap(System.Environment.CurrentDirectory +
3  "/" + "Documents" + @"\" + img1);
4  Bitmap bitmap2 =
5  new Bitmap(System.Environment.CurrentDirectory +
6  "/" + "Documents" + @"\" + img2);
7  Bitmap resized =
8  new Bitmap(bitmap1, bitmap2.Width, bitmap2.Height);
9  Bitmap copy = new Bitmap(resized.Width, resized.Height,
10 System.Drawing.Imaging.PixelFormat.Format24bppRgb);
11 using (System.Drawing.Graphics gr =
12 System.Drawing.Graphics.FromImage(copy))
13 {
14   gr.DrawImage(resized, new System.Drawing.Rectangle(
15   0, 0, copy.Width, copy.Height));
16 }
17 resized = copy;
```

Codebeispiel 6.9 Anpassen der Bilder

In **Codebeispiel 6.9** Anpassen der Bilder, **Zeile 1-6** ist zu sehen, wie die zu vergleichenden Bilder eingelesen werden.

In den **Zeilen 7-8** wird die Größe des ersten Bildes der Größe des zweiten Bildes angepasst. In **Zeile 10** wird als Pixelformat "Format24bppRgb" festgelegt, da die AForge.Imaging Bibliothek nur 24 bpp Bilder unterstützt.

6.4. Installer für das WPF Programm

Der Installer für die Lernhilfe dieser Diplomarbeit wurde ebenfalls in C#, genauer gesagt WPF, implementiert. Dieser dient dazu, die Lernhilfe sowie die benötigten Ressourcen (z.B. der Inhalt der Lernhilfe in Form von HTML-Dateien) in das korrekte Verzeichnis zu kopieren.

Dazu bedient sich der Installer an einem Ordner namens "installres", der alle Dateien, die zum korrekten Ablauf von MLA nötig sind, beinhaltet.

Wichtig hierbei ist es vor allem, die zu installierenden Dateien in den richtigen Zielordner zu kopieren. Folgender Code behandelt dieses Problem:

```
1  string SourcePath = System.Environment.CurrentDirectory
2  + "/installres";
3  string DestinationPath =
4  Environment.GetEnvironmentVariable("MLAB_ROOT") + "\\";
5  string[] subdirs = Directory.GetDirectories(
6  SourcePath, "*", SearchOption.AllDirectories);
7  for (int i = 0; i < subdirs.Count(); i++){
8    Directory.CreateDirectory(
9    subdirs[i].Replace(SourcePath, DestinationPath));
10 }
11 string[] dest = Directory.GetFiles(
12 SourcePath, "*.*", SearchOption.AllDirectories);
13 int all = dest.Count();
14 for (int i = 0; i < dest.Count(); i++){
15   File.Copy(dest[i], dest[i].Replace(
16   SourcePath, DestinationPath), true);
17   (sender as BackgroundWorker).ReportProgress(
18   ((i+1)*100)/all);
19 }
20 Dispatcher.Invoke(new Action(() =>
21 {
22   lblState.Content = "Installation abgeschlossen.";
23   btnClose.IsEnabled = true;
24 }));
```

Codebeispiel 6.10 Installieren der Dateien

Die in **Codebeispiel 6.10** genannte Variable „*SourcePath*“ legt fest, von wo die Dateien kopiert werden. Die Variable „*DestinationPath*“, also der Ort, an den die Dateien kopiert werden sollen, wird mit einem Pfad belegt, der aus der Umgebungsvariable "*MLAB_ROOT*" gelesen wird. Die Umgebungsvariable "*MLAB_ROOT*" wird bei der Installation von MeVisLab festgelegt und garantiert dadurch, den Installationspfad von MeVisLab zu finden. Dieser Pfad

wird benötigt, da MLA in dem Unterverzeichnis *"/Packages/MLA/*" installiert werden muss, damit das ML Modul (siehe **Kapitel 6.1**) darauf zugreifen kann.

In Zeile 7-10 werden zuerst alle Unterverzeichnisse angelegt, in den Zeilen 11-19 werden danach alle Dateien in diese Unterverzeichnisse kopiert. In den Zeilen 17 und 18 wird ein "*BackgroundWorker*" benutzt, damit dem Benutzer mitgeteilt werden kann, wie weit der Fortschritt der Installation bereits verlaufen ist. Da das Ganze in einem eigenen Thread aufgerufen wird, muss in den Zeilen 20-24 durch den Aufruf "*Dispatcher.Invoke*" noch dem UI-Thread mitgeteilt werden, dass die Installation fertig ist.

7. Technische Daten

7.1. Systemanforderungen

In diesem Kapitel werden Mindestvoraussetzungen an die Hardware sowie an die Software aufgelistet. Da die Diplomarbeit ein Modul des Programms MeVisLab ist, werden zuerst die Anforderungen dessen herangezogen und dann um die Anforderungen der Diplomarbeit ergänzt.

7.1.1. Software

- Windows XP, Vista , 7 oder 8
- MeVisLab
- .NET 4.0
- Microsoft Office Word

7.1.2. Hardware

- Intel Pentium III
- 1 GB RAM
- Eine OpenGL kompatible Grafikkarte

[mevislab.de]

8. Beurteilung

In diesem Kapitel wird die Diplomarbeit kritisch betrachtet, Probleme und deren Lösung werden hervorgehoben und Verbesserungsvorschläge werden genannt.

8.1. Kritik

Uns ist vor allem die Zusammenarbeit mit den zwei Studenten der FH für Gesundheitsberufe gut gelungen. Termine wurden immer pünktlich eingehalten und die Kommunikation über Email und andere digitale Medien funktionierte auch einwandfrei.

8.2. Probleme

Hier werden Probleme erklärt, die im Laufe dieser Diplomarbeit aufgekommen sind.

8.2.1. Module anfertigen

Zu Beginn war es notwendig, das Programm „MeVisLab“ generell zu verstehen und zu benutzen. Erst versuchten wir vergebens ein Modul anzulegen. Erst nach langer Recherche fanden wir ein Tutorial zum Anlegen eines Moduls, welches wir erfolgreich absolvierten.

Ein weiteres großes Problem war es, die angelegten Module auf anderen Rechnern zum Laufen zu bringen, ohne das gesamte Modul neu anlegen zu müssen. Hierzu wäre eine nicht-kommerzielle Lizenz notwendig, welche aus finanz- und zeittechnischen Gründen nicht akquiriert werden konnte. Daher mussten wir uns um eine Lösung umsehen, MLA auch auf anderen MeVisLab Installationen einzufügen.

Dies war natürlich nur in der Debug Version von MeVisLab möglich, da nur über die Lizenz ein ohne Debug-Modus nutzbares Modul erstellt werden kann.

8.2.2. Einlesen neuer Anleitungen

Anfangs wollten wir die Word-Dokumente der Anleitungen direkt einlesen, jedoch ist der Aufbau dieser Dokumente so unleserlich und komplex, dass kein Muster und keine Zusammenhänge gefunden wurden. Daraufhin überlegten wir uns welche Alternativen zur Verfügung stehen. Unserer Meinung nach war die einzig sinnvolle Möglichkeit, die Word-Dokumente zuerst in HTML-Dateien umzuwandeln und anschließend einzulesen, da der entstandene HTML-Code deutlich verständlicher und strukturierter aussah. Ein weiterer Punkt, der hierbei Probleme verursachte, war die Tatsache, dass Microsoft Office Word bei der Konvertierung in HTML, die enthaltenen Bilder teilweise doppelt, manchmal dreifach oder auch nur einmal abspeicherte. Wir wollten klarerweise das qualitativ hochwertigste Bild für unser Programm verwenden. Dazu war im Endeffekt ein Algorithmus unter Verwendung einer externen Bibliothek unter Verwendung.

8.2.3. Verbesserungen

Um eventuell einen Performancegewinn zu erzielen, wäre sinnvoll die Anleitungen direkt als „*.docx*“ Dateien einzulesen anstatt sie vorher in „*.html*“ umzuwandeln. Dies würde vor allem das Problem mit den unterschiedlichen Bildern beseitigen.

Um WPF optimal einzusetzen, wäre es von Vorteil, wenn sich das DataBinding über XAML durch die ganze Applikation ziehen würde, denn so wird das Programm logisch abgegrenzt. Aktuell werden diverse User-Controls im Code-Behind befüllt.

Um eine sinnvolle Beziehungen zwischen den Anleitungen herzustellen, wäre es noch notwendig, einen Dokumentenverweis in der Applikation zu realisieren.

8.3. Resümee

Im Laufe der Diplomarbeit konnten wir einen relativ tiefen Blick in die Medizin und deren Praktiken, speziell in die Radiologietechnologie, werfen. Vor allem die MRT und CT konnten wir hautnah erleben, da wir bei diesen Untersuchungen live dabei waren bzw. selbst vom MRT durchgescannt wurden.

Des Weiteren haben wir gelernt mit externen Partnern in direkter Zusammenarbeit zu stehen, wobei beide Parteien voneinander abhängig sind.

Wir sind froh, dass wir mehrere, teilweise unbekannte, Schnittstellen zu einer Gesamtlösung zusammenfügen konnten. Besonders auf der Seite von MeVisLab mussten wir uns gut in die Materie einlesen und einen Weg finden, die Brücke zwischen dem Modul und dem WPF Programm zu schlagen.

Literaturverzeichnis

AFORGE.NET. 2008-2012. *http://www.aforgenet.com/framework/features/*. [online]. [Accessed 14 Apr 2015]. Available from World Wide Web: <http://www.aforgenet.com/framework/features/>

ANTWERPES, Frank, Timo FREYER, and Benjamin ABELS. *Doccheck Flexikon*. [online]. [Accessed 11 März 2015]. Available from World Wide Web: <http://flexikon.doccheck.com/de/Schnittbildverfahren>

Axel Kock Medienbüro für Gestaltung. [online]. [Accessed 11 März 2015]. Available from World Wide Web: <http://www.axelkock.de/images/thumbs/ct.jpg>

biolernen. [online]. [Accessed 2015]. Available from World Wide Web: <http://biolernen.de/BIO_AB/PDF_N/kernspin.htm>

BONN, Matthias. 1999-2000. *Robotik in der Medizin*. [online]. [Accessed 11 März 2015]. Available from World Wide Web: <http://www.matze-bonn.de/informatik/Seminar_CT.pdf>

BORGHOFF, Thorsten and Nico HÖLL. 2005. *Bildverarbeitung und Algorithmen Vortrag Segmentierung*. [online]. [Accessed 11 März 2015]. Available from World Wide Web: <http://www.gm.fh-koeln.de/~konen/WPF-BV/Segmentierung-SS05.doc>

BUCHBERGER, Michael and Gerald ZWETTLER. 2014. *Radiologische Bildverarbeitung*. [online].

CLARK, James and Steve DEROSE. 1999. *XML Path Language (XPath)*. [online].

de.wikipedia.org. [online]. [Accessed 11 März 2015]. Available from World Wide Web: <http://de.wikipedia.org/wiki/Voxel>

dicom.offis.de. 2013. [online]. [Accessed 11 März 2015]. Available from World Wide Web: <http://dicom.offis.de/dcmintro.php.de>

Einführung in WPF. [online]. [Accessed 11 März 2015]. Available from World Wide Web: <https://msdn.microsoft.com/de-de/library/aa970268%28v=vs.110%29.aspx>

First Steps with MeVisLab. 2003-2012. [online].

GEMEINSCHAFTSPRAXIS FÜR RADIOLOGIE. *Gemeinschaftspraxis für Radiologie*. [online]. [Accessed 2015]. Available from World Wide Web: <http://www.radiologie-sendlingerstrasse.de/untersuchungen_koll.htm>

htmlagilitypack.codeplex.com. 2012. [online]. [Accessed 11 März 2015]. Available from World Wide Web: <http://htmlagilitypack.codeplex.com/>

Image Processing Library (ML). [online]. [Accessed 11 März 2015]. Available from World Wide Web: <http://www.mevislab.de/mevislab/features/image-processing/>

KIRILLLOV, Andre. *code.google.com/p/aforge/*. [online]. [Accessed 15 Apr 2015]. Available from World Wide Web: <https://code.google.com/p/aforge/>

meduniewien. [online]. [Accessed 2015]. Available from World Wide Web: <http://www.meduniwien.ac.at/7tesla/was-ist-mrt/wie-funktioniert-mrt-physikalisch-betrachtet/>

MEVIS MEDICAL SOLUTIONS. 2015. *The ML Programming Guide - Programming Object-Oriented Image Processing with the MeVis Library*. [online]. Available from World Wide Web: <http://www.mevislab.de/docs/current/MeVisLab/Resources/Documentation/Publish/SDK/MLGuide.pdf>

mevislab.de. [online]. [Accessed 2015]. Available from World Wide Web: <http://www.mevislab.de/mevislab/system-requirements/>

Microsoft MSDN. [online]. [Accessed 2015]. Available from World Wide Web: <https://msdn.microsoft.com/en-us/library/windows/desktop/ms686331%28v=vs.85%29.aspx>

Microsoft MSDN. [online]. [Accessed 2015]. Available from World Wide Web: <https://msdn.microsoft.com/en-us/library/windows/desktop/ms684873%28v=vs.85%29.aspx>

msdn.microsoft.com. [online]. [Accessed 11 März 2015]. Available from World Wide Web: <https://i-msdn.sec.s-msft.com/dynimg/IC49037.png>

MUCH, Julian. 2005. *Computertomographie Hauptseminar Bildverarbeitung*. [online]. [Accessed 11 März 2015]. Available from World Wide Web: <http://www9.in.tum.de/seminare/ps.WS05.gdbv/ausarbeitungen/CT.pdf>

NEMA. 2015. *PS3.1: DICOM PS3.1 2015a - Introduction and Overview*. [online]. [Accessed 11 März 2015]. Available from World Wide Web: <http://dicom.nema.org/medical/dicom/current/output/pdf/part01.pdf>

PABST, Dr. med. Christoph. 2013. *Grundlagen der Magnetresonanztomographie*. [online].

PAULUS, Dietrich and Sahla BOUATTOUR. 2003. *Hauptseminar: Medizinische Bildverarbeitung*. [online].

PETERWITZ, Julia. 2006. *Grundlagen der Bildverarbeitung und Objekterkennung*. [online]. Available from World Wide Web: <http://www9.in.tum.de/seminare/hs.SS06.EAMA/material/01_ausarbeitung.pdf >

POSPICH, Sabrina. 2012-2013. *Angiographie*. [online]. [Accessed 11 März 2015]. Available from World Wide Web: <https://e3.physik.uni-dortmund.de/~suter/Seminare/MedPhys12/Vortraege/7_CT-Zusammenfassung.pdf>

radiologicum münchen. [online]. [Accessed 2015]. Available from World Wide Web: <http://www.radiologicum-muenchen.de/Kernspintomographie-MRT/FAQ>

RADIOLOGIE, Gemeinschaftspraxis für. *Gemeinschaftspraxis für Radiologie*. [online]. [Accessed 2015]. Available from World Wide Web: <http://www.radiologie-sendlingerstrasse.de/impressum.htm>

radiologie.de. 2013. [online]. [Accessed 11 März 2015]. Available from World Wide Web: <http://www.radiologie.de/untersuchungsmethoden-im-ueberblick/>

SHANTO, Asherd. 2015. *www.codeproject.com*. [online]. [Accessed 11 März 2015]. Available from World Wide Web: <http://www.codeproject.com/Articles/659019/Scraping-HTML-DOM-elements-using-HtmlAgilityPack-H>

Technische Universität Dresden. [online]. [Accessed 11 März 2015]. Available from World Wide Web: <http://www.mfd.mw.tu-dresden.de/mfd/index.php/forschung/tomographie#Doktorarbeiten>

VOGEL, Thomas, Wolfang REITH, and Ernst RUMMENY. 2011. *Diagnostische und interventionelle Radiologie*. Springer-Verlag.

Wikipedia. [online]. [Accessed 11 März 2015]. Available from World Wide Web: <http://de.wikipedia.org/wiki/Positronen-Emissions-Tomographie>

www.fallsammlung-radiologie.de. [online]. [Accessed 11 März 2015]. Available from World Wide Web: <http://www.fallsammlung-radiologie.de/ct_multislice.html>

www.idir.uniklinikum-jena.de. [online]. [Accessed 2015]. Available from World Wide Web: <http://www.idir.uniklinikum-jena.de/idir_media/kirad_multimedia/VL_Zahn/VLZahn_MRT.pdf>

www.mevislab.de. [online]. [Accessed 11 März 2015]. Available from World Wide Web: <http://www.mevislab.de/mevislab/features/>

www.uni-bonn-radiologie.de. [online]. [Accessed 11 Februar 2015]. Available from World Wide Web: <http://www.uni-bonn-radiologie.de/front_content.php?idart=368>

www.uni-bonn-radiologie.de. [online]. [Accessed 11 Februar 2015]. Available from World Wide Web: <http://www.uni-bonn-radiologie.de/front_content.php?idart=379>

www.visualstudio.com. 2015. [online]. [Accessed 11 März 2015]. Available from World Wide Web: <http://www.visualstudio.com/explore/features-overview-vs>

ZWETTLER, Gerald. 2013. *Bildverbesserung*. [online].