Hochschule Wismar
Fakultät für Ingenieurwissenschaften / MVU, FG CEA

Projekt A

HINWEIS:
Das Projekt basiert auf Grundlagen der Vorlesung "Bildverarbeitung in der Medizin" vom Institut of Medical Engineering, der Universität zu Lübeck

ENTWICKLUNG EINES MATLAB PROGRAMMS MIT GUI ZUR DARSTELLUNG MEDIZINISCHER INTENSITÄTSBILDER

Eingereicht am: 30. Mai 2013
von: Adrian Lühr
geboren am 31. August 1987
in Cuxhaven

Aufgabenstellung

Entwicklung eines MATLAB Programms mit GUI zur Darstellung medizinischer Intensitätsbilder (Grauwertbilder) auf Basis von Unterlagen des Instituts für Medizintechnik der Universität zu Lübeck.

Schwerpunkte:

- Einarbeitung in theoretische Grundlagen digitaler Bildinformationen
- Softwaretechnische Erprobung ausgewählter Grauwerttransformationen an Beispielen
- Einarbeitung in die GUI Programmierung mit MATLAB
- Umsetzung einer GUI-basierten MATLAB Applikation zur Darstellung medizinischer Intensitätsbilder gemäß der erweiterten Aufgabenspezifikation
- Test der Applikation

Die Arbeit ist nach den vorgegebenen Richtlinien anzufertigen. Neben der geforderten Anzahl schriftlicher Exemplare ist die Arbeit (Originaldateien & PDF-Dateien) sowie die entwickelte Software auf einem dokumentierten digitalen Datenträger abzugeben.

Betreuer und Prüfer: Prof. Dr.-Ing. T. Pawletta, FIW/MVU, FG CEA, HS Wismar

Inhalt

1 Einleitung

Diese Projektarbeit beschäftigt sich mit dem Thema der Bildverarbeitung, wobei es sich im Speziellen um die Darstellung medizinischer Intensitätsbilder handelt. Das Ziel dieser Arbeit ist es, eine GUI-Benutzeroberfläche mit Hilfe von MATLAB zu erstellen, die medizinische Bilddaten verarbeiten kann. Dabei soll es möglich sein, Bilddaten einzulesen und diese in optimierter Darstellung wieder auszugeben, um eine medizinische Auswertung möglich zu machen.

Im Vorfeld der GUI-Erstellung wird eine softwaretechnische Erprobung ausgewählter Grauwerttransformationen durchgeführt und eine Einarbeitung in die GUI-Programmierung mit MATLAB vorgenommen.

2 Einführung in theoretische Grundlagen digitaler Bildverarbeitung

2.1 Allgemeine Grundlagen

Für eine digitale Bildverarbeitung gibt es eine Vielzahl von Anwendungsmöglichkeiten. In der Industrie werden Bildinformationen häufig zur Qualitätskontrolle oder zur Identifizierung verschiedener Objekte benutzt. Im Multimediadesign hingegen werden Bilder oft manipuliert und mit speziellen Effekten ausgestatten, um eine gewünschte Wirkung zu erzielen. Obwohl Bildinformationen so unterschiedlich genutzt werden, basieren die Bildverarbeitungsmethoden alle auf den gleichen Grundlagen.

Ein Bild basiert nach [**1**] immer auf einer Matrix. Die Bildmatrix (z.B. in 2D) entspricht also der Funktion $f(x,y)$. Jedes Bild besitzt einen Wertebereich G, der die möglichen Grauabstufungen beinhaltet. Neben Grauwertintervallen gibt es auch Rot-, Grün-, und Blau- (RGB-) Intervalle, die aber in dieser Arbeit nicht behandelt werden. Der Grauwertbereich g eines Bildes befindet sich innerhalb des Wertebereichs und gibt die tatsächlich im Bild vorhandenen Grauwerte an.

Wertebereich: $$G = G_{min}, \dots, G_{max} \tag{2.1}$$

Grauwertbereich: $$g = g_{min}, \dots, g_{max} \in G \tag{2.2}$$

Aufbau der Bildmatrix: $$f(x,y) \in G^{n_x * n_y} \tag{2.3}$$

Wobei mit n die Anzahl der Bildpunkte in x- und y-Richtung Angegeben wird.

Bei einer Transformation T wird ein Quellbild f durch Transformation der Grauwerte so beeinflusst, dass ein Zielbild f‘ entsteht.

Transformation: $$f'(x,y) = T_{xy}(f(x,y)) \tag{2.4}$$

Alle in dieser Arbeit behandelten Transformationen sind homogene Punktoperationen. Das bedeutet, dass einem Grauwert g aus einem Quellbild gemäß der Transformationsgleichung T ein neuer Grauwert g‘ zugewiesen wird, ohne dass die Umgebung berücksichtigt wird.

2.2 Bilder in der Medizin

In der Medizin wird nach [**2**] für Bilddaten der DICOM Standard verwendet. DICOM steht für „Digital Imaging Communications in Medicine“ und beschreibt einen Standard digitaler Bildverarbeitung. DICOM ermöglicht eine barrierefreie Kommunikation verschiedener Institutionen über Ländergrenzen hinweg. Neben dem Dateiformat umschreibt DICOM auch die Speicherung, den Datentransfer und die Darstellung medizinischer Bilddaten. DICOM unterstützt bis zu 65536 Graustufen. Dies ermöglicht einen sehr großen Detailreichtum. Konventionelle Formate wie zum Beispiel JPEGs oder BMP Dateien unterstützen nur bis zu 256 Graustufen.

3 Softwaretechnische Erprobung ausgewählter Grauwerttransformationen

3.1 Lineare Histogrammspreizung

Die Lineare Histogrammspreizung ist ein Verfahren zur Verstärkung des subjektiv wahrgenommen Bildkontrastes. Häufig wirken Bilder kontrastarm, weil der Wertebereich nur zum Teil vom vorhandenen Grauwertbereich ausgenutzt wird. Das bedeutet, dass nur ein Teil der zur Verfügung stehenden Grauabstufungen genutzt wird. Zur subjektiven Verbesserung des Bildkontrastes wird der genutzte Grauwertbereich auf den Wertebereich ausgendehnt.

Die Grauwerttransformation wird durch die Gleichung 3.1 beschrieben.

Histogrammspreizung:

$$T_{stretch}(g) = \frac{g'_{max} - g'_{min}}{g_{max} - g_{min}} * (g - g_{min}) + g'_{min} \qquad (3.1)$$

In MATLAB kann durch den *imshow* Befehl eine Histogrammspreizung durchgeführt werden. Wird ein Bild über *imshow* dargestellt, benutzt MATLAB automatisch den Wertebereich des Datentyps als Wertebereich G. Bei dem Datentyp *int16* wäre das zum Beispiel das Intervall [-32768 32767]. Dies führt in der Regel nicht zu zufriedenstellenden Ergebnissen, da grade durch die kurzen Belichtungszeiten in der Medizin (die der Strahlungsminderung dienen) nicht korrekt belichtete Bilder vorhanden sind. Um die Bildausgabe zu optimieren, kann dem Befehl *imshow* der Grauwertbereich des Bildes als Parameter übergeben werden. MATLAB gleicht so den Wertebereich an den Grauwertbereich an und führt eine lineare Histogrammspreizung durch. Unabhängig von der Größe des Wertebereiches eines Bildes verteilt die *imshow* Funktion 256 Grauwerte gleichmäßig über das Intervall. Dies wird so gehandhabt, da gewöhnliche Monitore nicht mehr als 256 Graustufen darstellen können.

Zur Veranschaulichung dieses Vorganges wird eine lineare Histogrammspreizung an einem Beispielbild vorgenommen.

Das in Abbildung 1 dargestellte Beispielbild ist ein axiales CT-Schnittbild des Kopfes vom Datentyp *int16*. Der Wertebereich einer *int16* Datei ist G= [-32768 32767]. Der in der Datei vorhandene Grauwertbereich beträgt aber nur g= [-1024 1962]. Dementsprechend wird ein großer Teil der verfügbaren Grauwerte nicht verwendet.

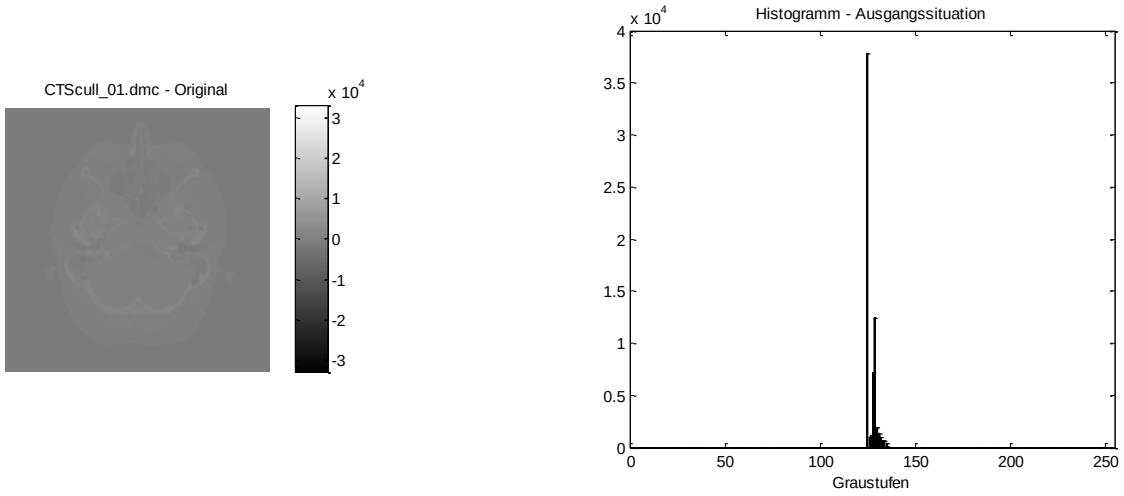


Abbildung 1: CT- Aufnahme mit zugehörigem Histogramm

Im Schnittbild sind nur schemenhaft Strukturen zu erkennen, was das Bild für eine medizinische Auswertung untauglich macht. Das Histogramm zeigt deutlich, dass nur ein kleiner Teil des Wertebereichs genutzt wird. Zur Kontrastverbesserung wird eine Histogrammspreizung nach Formel 3.1 durchgeführt.

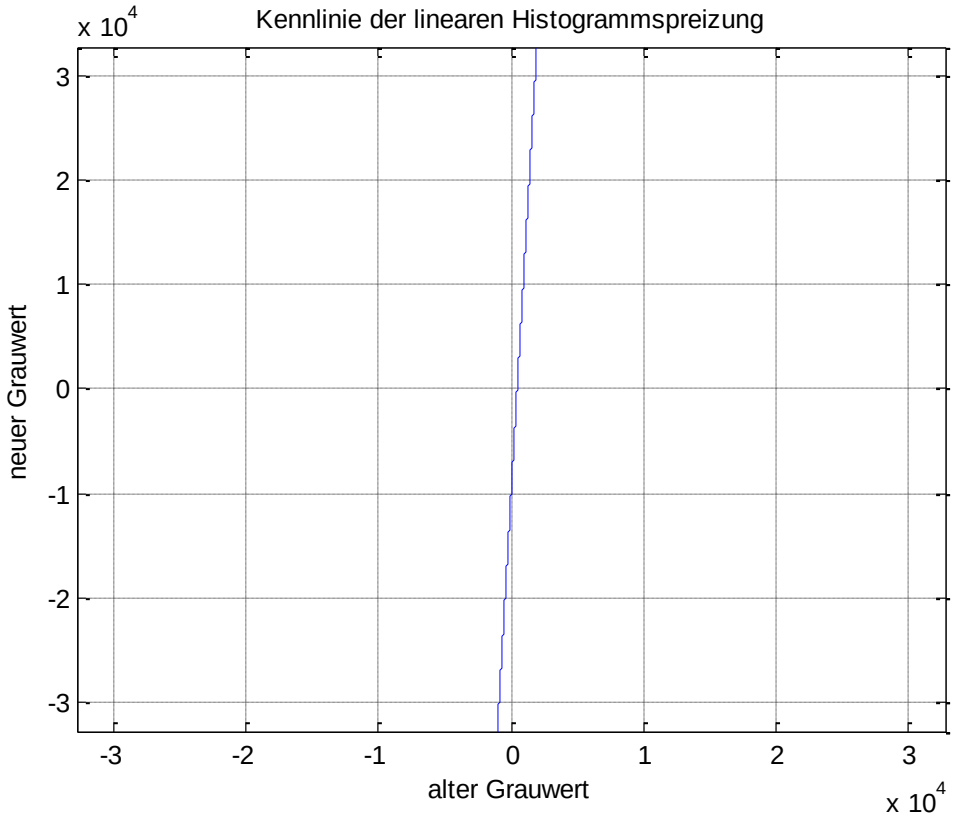


Abbildung 2: Kennlinie der linearen Histogrammspreizung

Die Transformationskennlinie zeigt die Abbildung der alten Grauwerte g des Quellbildes auf die Grauwerte g‘ des Zielbildes. Hierbei ist deutlich zu erkennen, dass der Grauwertbereich des Quellbildes sehr schmal ist.

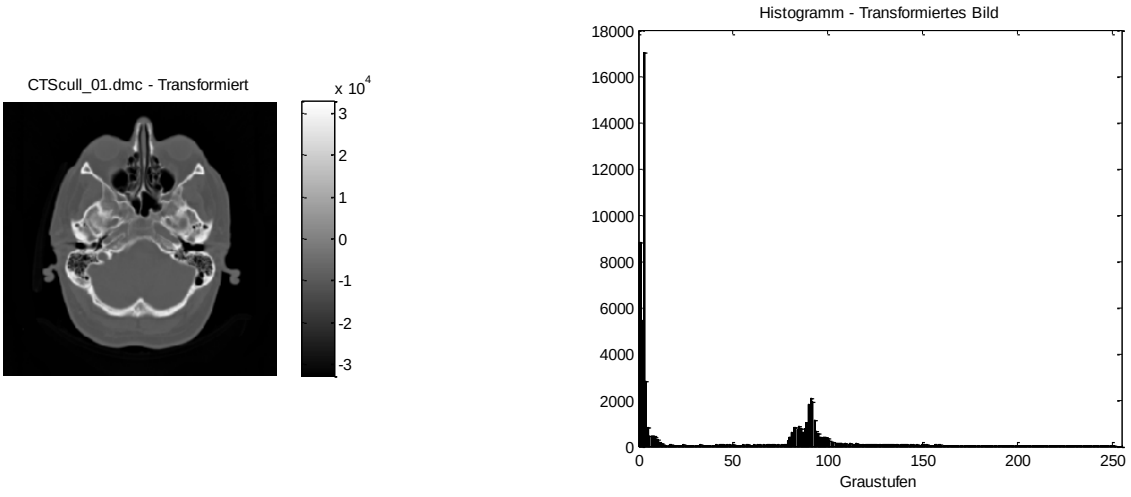


Abbildung 3: Transformierte CT- Aufnahme mit zugehörigem Histogramm

Nach der Transformierung ist das Bild kontrastreicher und es lassen sich auch feine Strukturen erkennen. In diesem Zustand wäre eine medizinische Auswertung möglich.

3.2 Window- Leveling

Das Window- Leveling oder auch Fensterung wird zur Darstellung von CT-Bilddaten verwendet, die einen sehr hohen Informationsgehalt beinhalten. Eine CT- Aufnahme kann nach [**3**] bis zu 4096 Grauwerte enthalten, während gewöhnliche Bildschirme nur 256 Graustufen darstellen können. Um ein solches CT- Bild medizinisch auswerten zu können, werden sogenannte Fenster definiert, in denen das darzustellende Gewebe kontrastreich abgebildet wird. Die Transformation des Bildes erfolgt durch die Angabe der Fensterbreite und des Fenstermittelpunktes. Alle Originalgrauwerte die außerhalb des gewählten Fensters liegen, werden abhängig von ihrem Wert auf 0 (schwarz) oder 255 (weiß) gesetzt.

Die Formel 3.2 beschreibt die Transformationskennlinie des Window- Leveling.

Window- Leveling:

$$T_{wl}(g) = \begin{cases} g'_{min} & \text{für } g < g_{min} \\ g'_{max} & \text{für } g > g_{max} \\ T_{stretch}(g) & sonst \end{cases} \qquad (3.2)$$

Das CT- Verfahren basiert nach [**4**] auf der Hounsfield- Skala, die den Zusammenhang von Dichteunterschieden und Grauwerten beschreibt. Aus diesem Zusammenhang lassen sich direkte Rückschlüsse auf das zugrundeliegende Gewebe schließen.

Daraus ergeben sich folgende Fensterungen:

- Knochenfenster: width= 1500 level= 300
- Lungenfenster: width= 2000 level= -200
- Weichteilfenster: width= 350 level= 50
- Gehirnfenster: width= 100 level= 35

Zur Veranschaulichung wird das Window- Leveling an einem Beispiel durchgeführt. Als Ausgangsbild dient wie in Kapitel 3.1 das axiale CT-Schnittbild des Kopfes. In Abbildung 4 sind die verschiedenen Fensterungen abgebildet. Die Verbesserung des qualitativen Kontrastes ist klar erkennbar. Das Knochenfenster stellt die Strukturen des Schädelknochens scharf und

kontrastreich dar, während das übrige Bild kontrastarm ist. Beim Weichteil- und Lungenfenster ist das Verhalten äquivalent und stellt das gewünschte Gewebe klar und kontrastreich dar. Das Lungenfenster führt bei diesem Beispielbild erwartungsgemäß zu keiner expliziten Bildverbesserung, da im Bild kein Lungengewebe vorhanden ist.

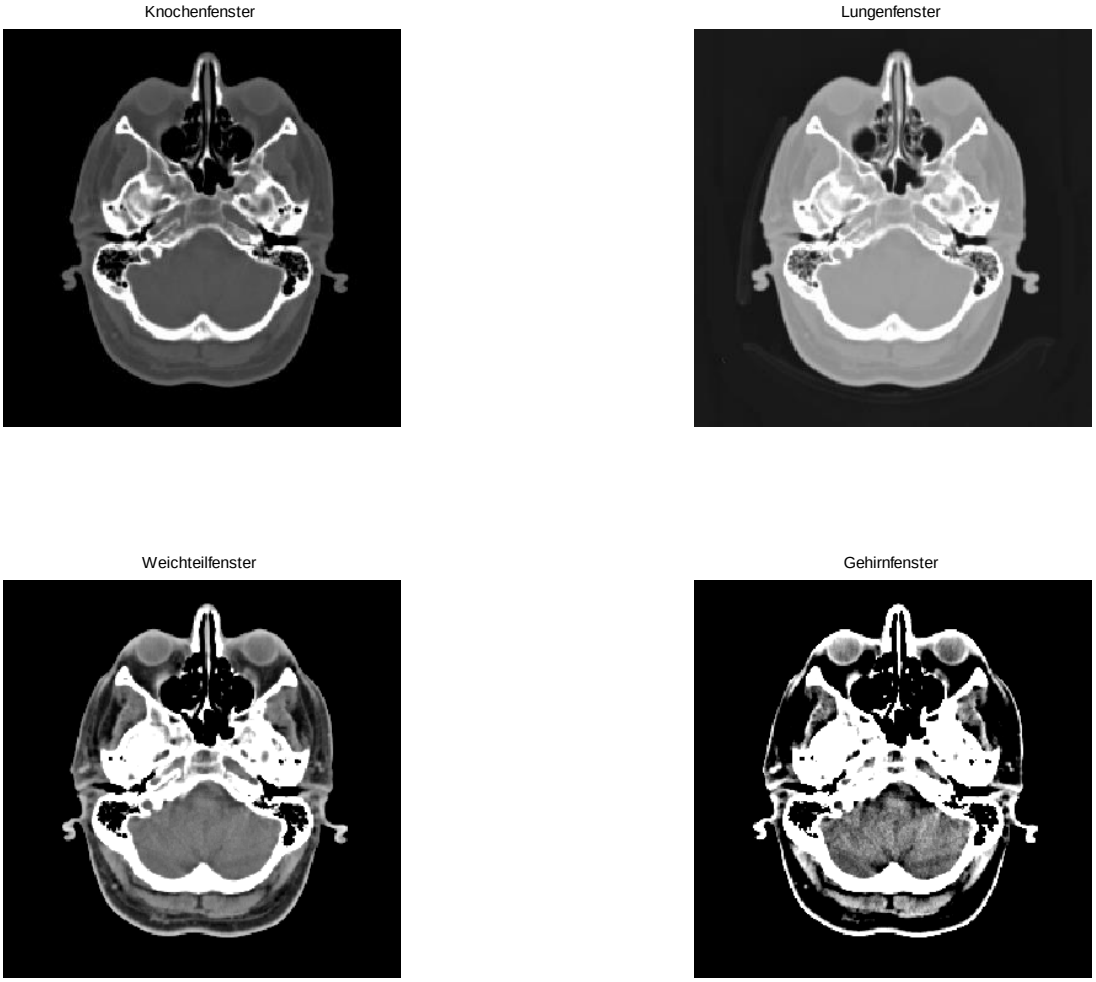


Abbildung 4: Darstellung der verschiedenen Fensterungen

3.3 Histogrammäqualisation

Die Histogrammäqualisation oder auch Histogrammausgleich genannt ist ein weiteres Verfahren zur Kontrastverbesserung von Grauwertbildern. Ziel dieses Verfahrens ist es, aus dem Histogramm des Quellbildes eine Gleichverteilung zu erzeugen, um den Wertebereich der Bilddatei ausnutzen zu können. Im Gegensatz zu dem zuvor behandelten Window Leveling gehen hier keine Bildinformationen verloren. Anhäufungen von Grauwerten werden gespreizt und

wenig bzw. nicht genutzte Bereiche werden gestaucht, aber nicht abgeschnitten. Die Transformationskennlinie wird mit der folgenden Formel beschrieben.

Histogrammäqualisation:

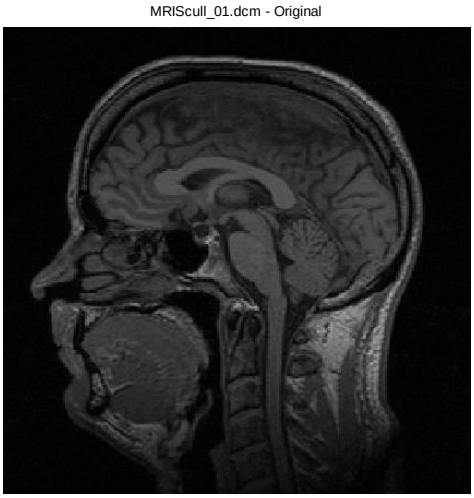
$$T_{equal}(g) = (g'_{max} - g'_{min}) \cdot \sum_{\xi=g_{min}}^{g} p(\xi) + g'_{min} \qquad (3.3)$$

Durch einen Histogrammausgleich endstehen im Ergebnisbild „Lücken“. Diese Lücken sind darin begründet, weil Grauwerte stets diskret vorliegen. Ein diskreter Grauwert kann nur auf einen anderen diskreten Grauwert umverteilet werden und nicht gespreizt werden.

In MATLAB kann eine Histogrammäqualisation über den Befehl *histeq* durchgeführt werden. Zusätzlich zum Quellbild können der Funktion *histeq* auch weitere Parameter übergeben werden.

Zur besseren Darstellung wird die Histogrammäqualisation an einem Beispiel durchgeführt. Als Ausgangsbild dient ein MRT des Kopfes in sagitaler Schnittebene.

Da das MRT- Bild ein Grauwertintervall von [0 179] besitzt, ist der Datentyp *int16* überdimensioniert und wurde über die Funktion *cast* zu *uint8* konvertiert.



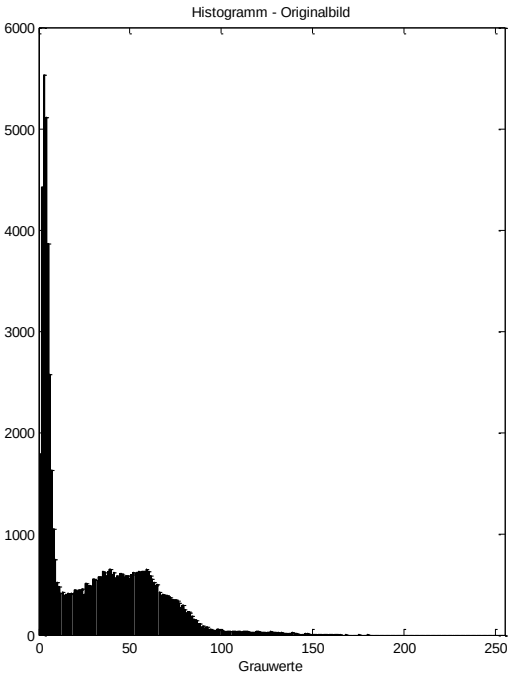


Abbildung 5: Ausgangsbild mit Histogramm

Abbildung 5 zeigt das unveränderte Ausgangsbild. Das Bild ist sehr dunkel und im Histogramm ist zu sehen, dass nur ungefähr die Hälfte des Wertebereichs genutzt wird. Über die Funktion *histeq* wird nun eine Histogrammäqualisation durchgeführt.

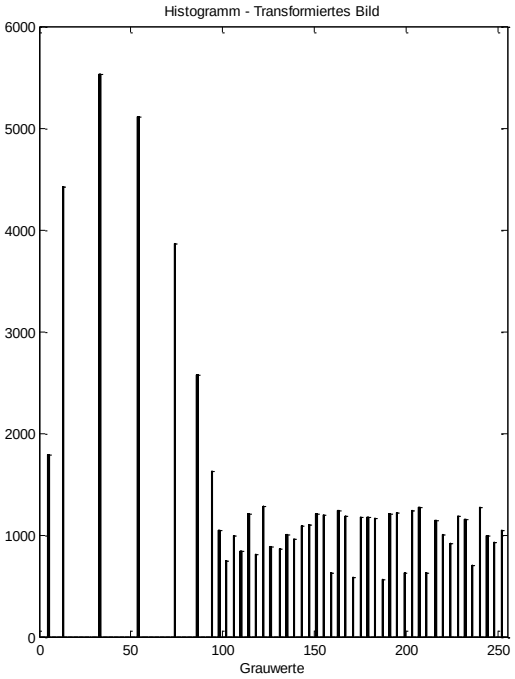


Abbildung 6: Transformiertes Bild mit Histogramm

Wie in Abbildung 6 zu sehen ist, wird das Bild jetzt deutlich heller dargestellt.

Das Histogramm zeigt, dass nun der gesamte Wertebereich genutzt wird. Des Weiteren sind die aus der Gleichverteilung resultierenden Lücken klar zu erkennen, die sich im Bild als weiße Punkte bemerkbar machen.

3.4 Histogrammhyperbolisation

Die durch Histogrammäqualitisation entstehenden Bilder wirken für das menschliche Auge oft zu hell, weil das menschliche Auge eine logarithmische Intensitätsempfindung besitzt. Bei der Histogrammhyperbolisation wird der menschlichen Wahrnehmung entgegengewirkt, indem der Gleichverteilung ein hyperbolischer Charakter verliehen wird. Die Formel für die Histogrammhyperbolisation ergibt sich aus Formel 3.3, die um einen Exponenten erweitert wurde.

Histogrammhyperbolisation:

$$T_{hyperb}(g) = \left(\left({g'_{max}}^{(\alpha+1)} - {g'_{min}}^{(\alpha+1)} \right) \cdot \left(\sum_{\xi=g_{min}}^{g} p(\xi) \right) + {g'_{min}}^{(\alpha+1)} \right)^{\frac{1}{\alpha+1}} \quad (3.4)$$

Der Parameter α kann Werte im Intervall -1 < α ≤ 0 annehmen und bestimmt damit den hyperbolischen Charakter der Transformationskennlinie. Wird α=0 gewählt, entspricht die Histogrammhyperbolisation der Histogrammäqualitisation. Da diese Transformation in dieser Arbeit nur mit Bilddaten vom Typ uint8 vorgenommen wird, wodurch sich ein Wertebereich von G=[0 255] ergibt, kann die Formel vereinfacht werden. Für g_{min}=0 und g_{max}=255 ergibt sich:

$$T_{hyperb}(g) = (G-1) \cdot \left(\sum_{\xi=g_{min}}^{g} p(\xi) \right)^{\frac{1}{\alpha+1}} \quad (3.5)$$

Über ein MATLAB- Script wurde diese Transformation an einem Beispielbild mit verschiedenen Werten für α vorgenommen. Für α wurde eingesetzt: 0,-1/3, -1/2,

-2/3, -0.8. Als Beispielbild dient wie in Kapitel 3.3 auch der Sagitalschnitt eines Kopfes. Zur effizienteren Berechnung der neuen Grauwerte wird eine Lookup-Tabelle (LUT) genutzt. In der LUT werden die möglichen Grauwerte laut Wertebereich, die absolute Häufigkeit der im Quellbild vorkommenden Grauwerte und die mit der Formel 3.5 berechneten Grauwerte gespeichert. Durch dieses Vorgehen muss bei der eigentlichen Transformation nur in der LUT nachgeschlagen werden, wodurch nur einmal jeder mögliche Grauwert berechnet werden muss und Rechenzeit gespart wird.

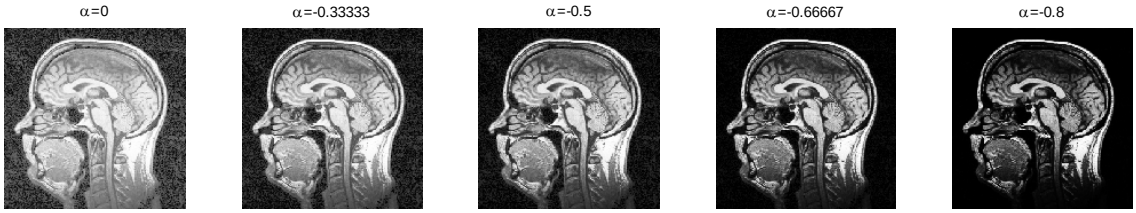


Abbildung 7: Einfluss des Parameters α

Bei einem Parameter von α= -0.8 ergibt sich folgende Transformationskennlinie. Durch den groß gewählten Parameter α= -0.8 ist das hyperbolische Verhalten klar ersichtlich.

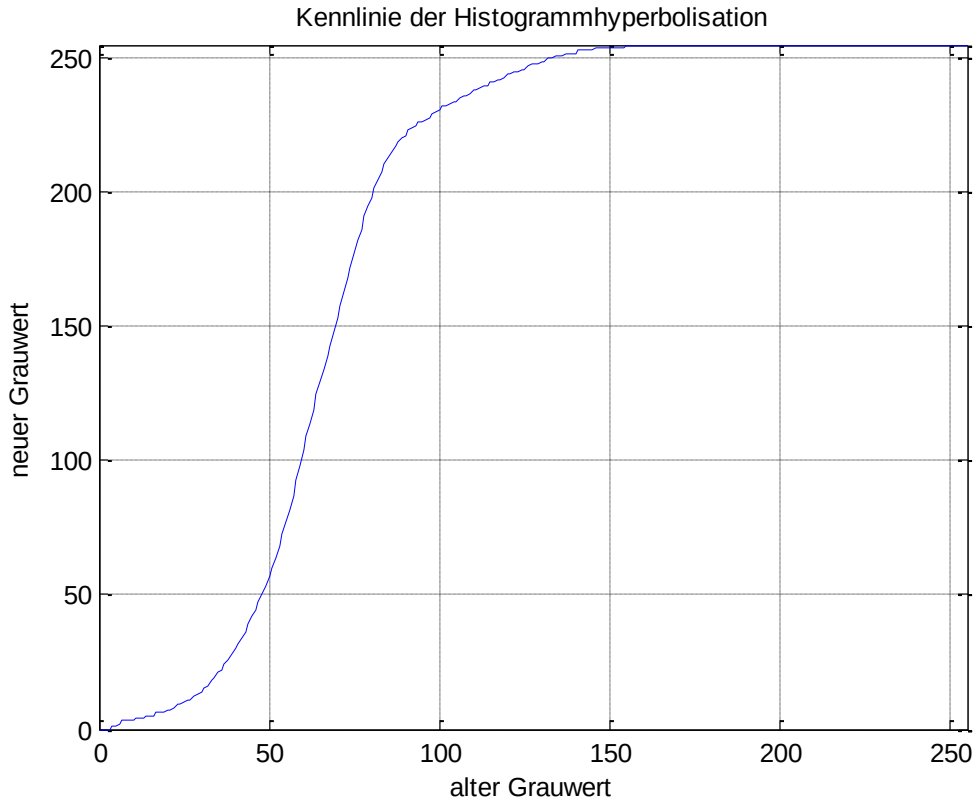


Abbildung 8: Kennlinie der Histogrammhyperbolisation (α = -0.8)

3.5 Bewertung der Grauwerttransformationen

Alle vier Transformationen dienen der Bildverbesserung und verbessern damit die medizinische Auswertbarkeit der Bilder. Die lineare Histogrammspreizung eignet sich für CT- und MRT- Bilder gleichermaßen, weil es bei beiden Bildtypen vorkommen kann, dass der Wertebereich G nicht voll ausgenutzt wird. Das Window- Leveling hingegen eignet sich ausschließlich für CT- Bilder. Die Grauwerte eines CT- Bildes stehen nach [**4**] über die Hounsfield- Skala in direktem Zusammenhang mit dem abgebildeten Gewebe. Der Grauwertbereich einer CT- Aufnahme erstreckt sich in der Regel über 12 Bit und umschließt damit das Intervall [-1024 3071]. Um diesen Grauwertbereich abbilden zu können, ist es nötig den Datentyp *int16* zu benutzen, der das Intervall [-32768 32767] einschließt, obwohl dieser nur zu einem Teil genutzt wird. Die Informationen die sich nicht in dem ausgewählten Fenster befinden, gehen beim Window- Leveling verloren. Ein Histogrammausgleich oder eine Histogrammhyperbolisation hingegen eignen sich in der hier dargestellten Form ausschließlich für MRT- Aufnahmen, weil sie für einen Wertebereich G= [0 255] ausgelegt sind. Die vorliegende MRT- Aufnahme besitzt einen Grauwertbereich g=[0 179] was eine Typenumwandlung (casting) vom Datentyp *int16* zu *uint8* zulässt. Durch das *casting* wird zusätzlich nicht benötigter Speicherplatz eingespart.

Alle hier bearbeiteten Transformationen sind irreversibel. Es ist also nicht möglich aus einem transformierten Bild wieder das Originalbild zu errechnen. Um diesem Problem entgegen zu wirken, wurden alle Funktionen so angelegt, dass das Rohmaterial lediglich gelesen wurde. So bleibt die Originaldatei immer unverändert und es können verschiedene Transformationen nacheinander auf ein Bild angewendet werden.

4 Umsetzung einer GUI- basierten MATLAB- Applikation

Die zu entwickelnde MATLAB- Applikation soll es ermöglichen, ausgewählte Grauwerttransformationen an medizinischen Intensitätsbildern durchzuführen. Die grafische Benutzeroberfläche (GUI) soll durch eine klare Gliederung und einen einfachen Aufbau auch Personen ohne umfassendes Hintergrundwissen über Grauwerttransformationen den Umgang mit medizinischem Bildmaterial ermöglichen. Die durch verschiedene Grauwerttransformationen erreichbare optimierte Darstellung von CT- und MRT- Bildern ermöglicht das Herausstellen verschiedener Strukturen und damit eine Grundlage zur medizinischen Auswertung.

4.1 Programmstruktur und Implementierung

Als Basis zur Entwicklung des Programmes dienten die zuvor entwickelten und in Kapitel 2 beschrieben MATLAB- Skripte zu den verschiedenen Grauwerttransformationen.

Die MATLAB- Skripte wurden den Anforderungen der Applikation angepasst und zu Funktionen umgewandelt. Die daraus resultierenden Funktionen sind:

hist_stretch.m → Lineare Histogrammspreizung

window_level.m → Window- Leveling

hist_eq.m → Histogrammausgleich

hist_hyp.m → Histogrammhyperbolisation

Durch dieses Vorgehen wird eine übersichtliche Struktur geschaffen. Diese Struktur sorgt für eine klare Trennung zwischen den verschiedenen Transformationen, gewährleistet eine gute Wartungsmöglichkeit und erleichtert Erweiterungen.

Die Funktion *grayValueTransformations_GUI.m* und die Layout- Datei *grayValueTransformations_GUI.fig* bilden den Kern des Programms. Über die Funktion *grayValueTransformations_GUI.m* wird das GUI und die

Unterfunktionen der verschiedenen Grauwerttransformationen gesteuert.

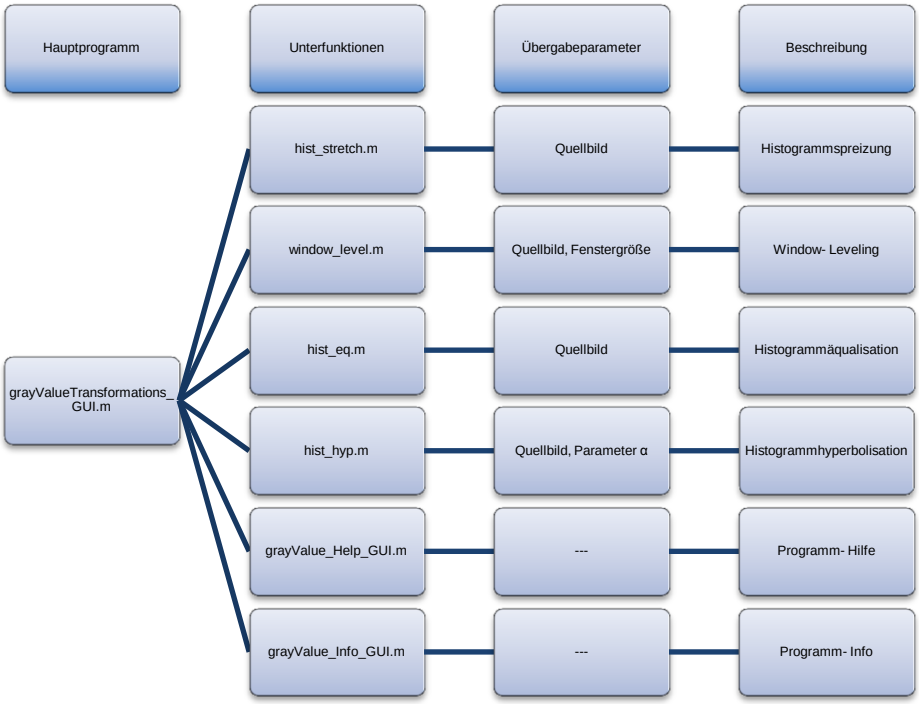


Abbildung 9: Elementarer Aufbau der Applikation

Die grafische Benutzeroberfläche wurde mit dem MATLAB Tool GUIDE (Graphical User Interface Design Environment) erstellt. Dieses umfangreiche Werkzeug ermöglicht es dem Anwender, komplexe Benutzeroberflächen zu gestalten. Zu diesen Benutzeroberflächen wird automatisch der MATLAB- Code generiert. Ein Beispiel für einen automatisch generierten MATLAB- Code wäre eine Callback- Funktion.

```
% --- Executes on button press in lineare_hist_button.
function lineare_hist_button_Callback(hObject, eventdata, handles)
% hObject    handle to lineare_hist_button (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
hist_stretch(handles.image_source)
```

Abbildung 10: Beispiel einer Callback- Funktion

In Abbildung 8 wird ein Beispiel einer automatisch erstellten Callback- Funktion dargestellt. Diese Funktion gehört zu einer Schaltfläche und wird durch anwählen der Schaltfläche ausgeführt.

4.2 Benutzeroberfläche der Applikation

Nach dem Start der Funktion *grayValueTransformations_GUI.m* erscheint die Programmoberfläche gemäß Abbildung 11.

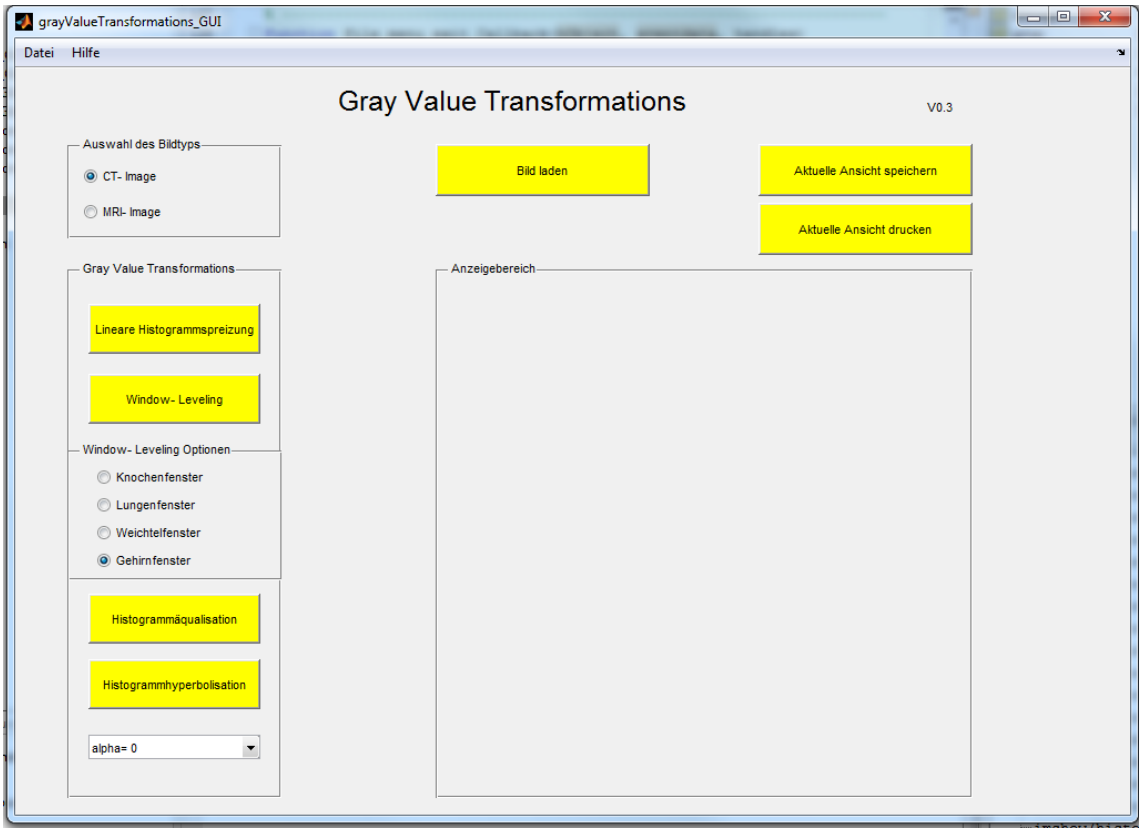


Abbildung 11: Benutzeroberfläche nach dem Programmstart

Zur Steigerung der Benutzerfreundlichkeit wurde wie in Abbildung 11 zu sehen, eine Farbcodierung der Schaltflächen eingeführt. Die Farbgebung der Schaltflächen ist abhängig davon, ob und welche Art von Bild geladen ist. Realisiert wurde diese Funktion durch ein *if- elseif-* Konstrukt, dass auf die *radiobuttons* im Bereich „Auswahl des Bildtyps“ zurückgreift. Diese Funktion ermöglicht es dem Nutzer visuell zu erfassen, welche Transformationen mit dem aktuell geladenen Bild durchführbar sind.

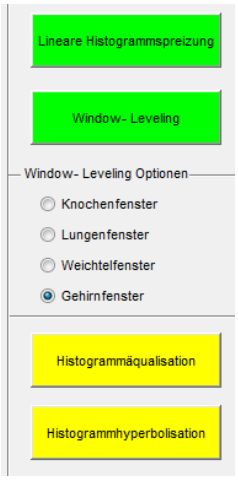


Abbildung 12: Visuelle Nutzerführung bei einem CT-Bild

Bevor ein Bild geladen wird, werden alle Schaltflächen gelb eingefärbt. Wenn ein Bild geladen wurde, wechseln die mit diesem Bild kompatiblen Transformationen auf grün.

Über die Schaltflächen im Bereich „Gray Value Transformations“ werden die zu den Grauwerttransformationen gehörenden Unterfunktionen aufgerufen. Als Parameter wird den Funktionen das vom Nutzer geladene Bild übergeben. Beim Window- Leveling und bei der Histogrammyperbolisation werden noch zusätzliche Parameter übergeben die über *radiobuttons* oder ein Dropdown-Menü wählbar sind. Bei dem Window- Leveling wurde wie bei der Farbgebung auch ein *if- elseif-* Konstrukt eingesetzt. Bei der Histogrammhyperbolisation dagegen wurde die *switch* Funktion eingesetzt, um den Zusammenhang zwischen Dropdown- Menü und dem Wert für α herzustellen.

Wurde ein Bild geladen, wird es im „Anzeigebereich“ dargestellt.

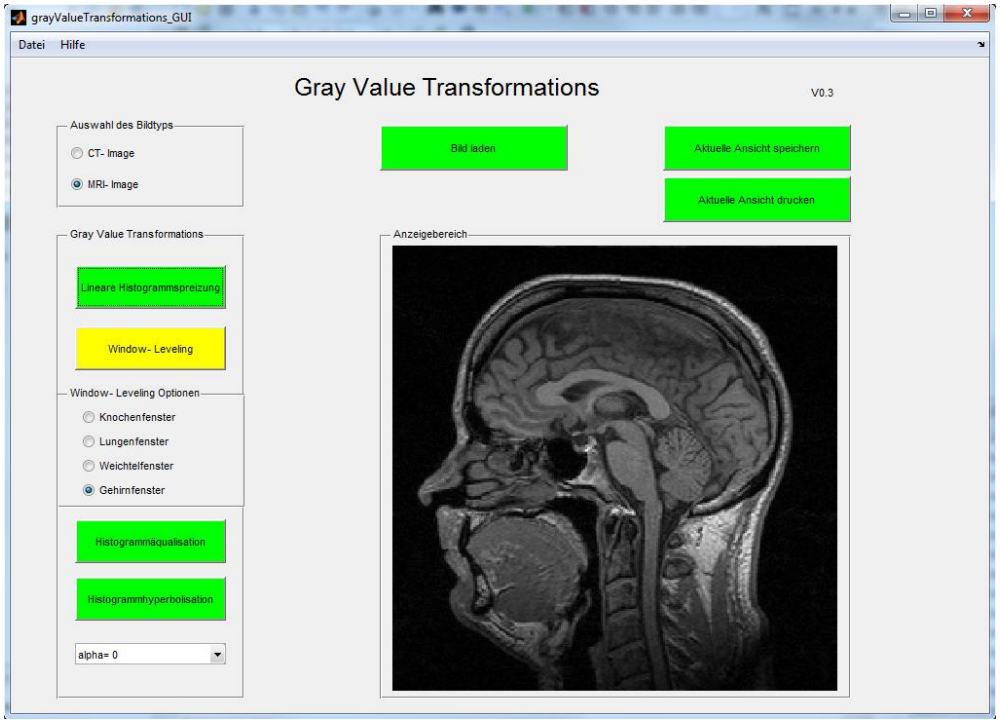


Abbildung 13: GUI mit geladener MRT- Aufnahme

Über die „Aktuelle Ansicht speichern“- sowie „Aktuelle Ansicht drucken“-Schaltflächen kann das aktuell angezeigte Bild gespeichert bzw. gedruckt werden. Diese Funktionalität wurde über die Funktion *getframe(handles.axes1)* realisiert. Die Funktion *getframe* ermöglicht es, nur das reine Bild ohne GUI abzuspeichern bzw. zu drucken.

Zur zusätzlichen Bedienerfreundlichkeit wurde eine Menüleiste implementiert über die das GUI beendet werden kann und eine Programmhilfe aufgerufen werden kann.

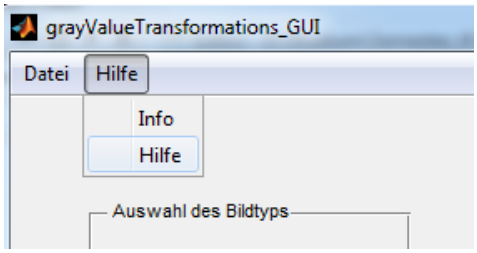


Abbildung 14: Menüleiste mit Programmhilfe

5 Nachweis der Funktionsfähigkeit und Bewertung der Lösung

Die Funktionsfähigkeit der Applikation wurde ausgiebig anhand von Beispielbildern getestet. Es wurden die Ergebnisse der Applikation mit den Ergebnissen der zuvor erstellten einzelnen Funktionen verglichen. Des Weiteren wurden die Transformationsergebnisse mit denen aus dem gegebenen Vorlesungsmaterial der Universität Lübeck [**5**] verglichen.

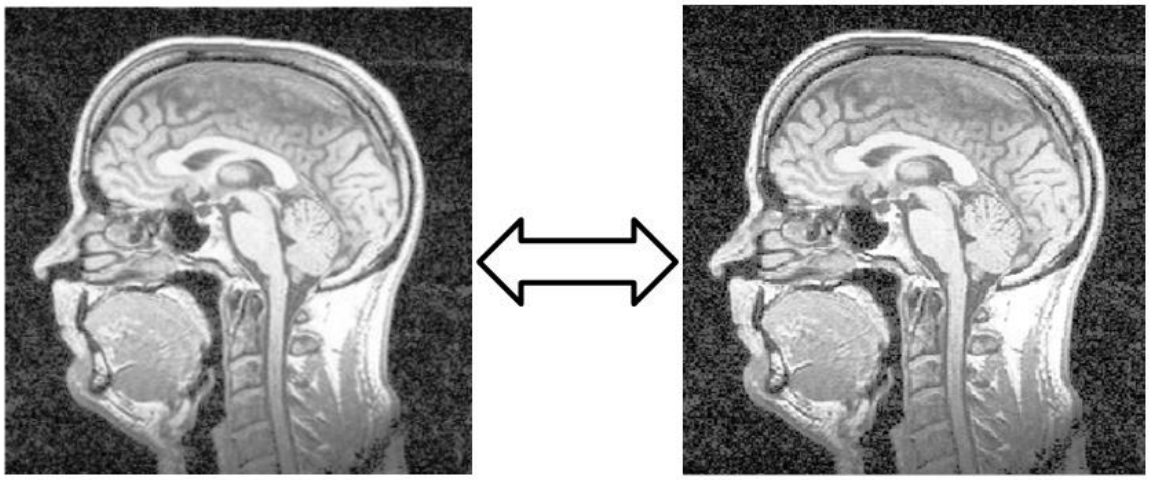


Abbildung 15: Vergleich der Transformationsergebnisse

Die Gegenüberstellung der transformierten Bilder hat in allen Fällen eine Übereinstimmung gezeigt. In Abbildung 13 und 14 ist als Beispiel die Gegenüberstellung eines MRT- Bildes, auf welches die Histogrammäqualisation angewendet worden ist, zu sehen.

Zusätzlich zu der Funktion der Grauwerttransformationen wurde auch die Stabilität der Applikation getestet. Hierzu wurden bewusst Fehleingaben getätigt, die aber durch Fehlermeldungen abgefangen wurden.

Nach meiner Einschätzung ist mit der Applikation ein stabiles und leicht zu bedienendes Programm entstanden.

6 Zusammenfassung und Ausblick

In Laufe dieser Projektarbeit ist ein benutzerfreundliches Programm zur optimierten Darstellung von medizinischen Intensitätsbildern entstanden. Durch die klare Gliederung der grafischen Oberfläche und die visuelle Nutzerführung ist es auch für Personen ohne größeres Hintergrundwissen möglich, medizinisches Bildmaterial in einer auswertbaren Form darzustellen. Durch die Aufteilung der Grauwerttransformationen in voneinander unabhängige Unterfunktionen wird eine einfache Erweiterbarkeit der jeweiligen Unterfunktion ermöglicht. Darüber hinaus kann der Funktionsumfang der gesamten Applikation ohne viel Aufwand durch Hinzufügen weiterer Grauwerttransformationsfunktionen erweitert werden.

Zur Verbesserung des Programmes könnten neben homogenen Punktoperationen auch Bereichsoperationen eingefügt werden. Des Weiteren würden Funktionalitäten zur Ausschnittsvergrößerung den Funktionsumfang und den medizinischen Nutzen der Applikation um ein Vielfaches erhöhen.

Literaturverzeichnis

[1] Thorsten Pawletta, *Aufgabenstellung*. Wismar, 2013.

[2] Oleg Pianykh, *Digital Imaging and Communications in Medicine (DICOM)*. Heidelberg: Springer-Verlag, 2008.

[3] Mikhail Gevantmakher and Christoph Meinel, *Medizinische Bildverarbeitung - eine Übersicht [Online]*. Universität Trier, http://www.math.uni-trier.de/verschiedenes/forschung/04_03.pdf (24.05.2013).

[4] Karl-Friedrich Masuhr and Marianne Neumann, *Neurologie: 128 Tabellen*, 6th ed. Stuttgart, New York: Thieme, 2007.

[5] Institut für Medizintechnik, *Bildverarbeitung in der Medizin*. Universität Lübeck.

Abbildungsverzeichnis

Abkürzungsverzeichnis

BMP	Bitmap
CT	Computertomographie
DICOM	Digital Imaging and Communications in Medicine
GUI	Graphical User Interface (Grafische Benutzeroberfläche)
GUIDE	Graphical User Interface Design Environment
JPEG	Joint Photographic Experts Group (Dateiformat)
LUT	Lookup- Tabelle
MRT	Magnetresonanztomographie
RGB	Rot Grün Blau

Anlagenverzeichnis

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die hier vorliegende Arbeit mit dem Thema „Entwicklung eines MATLAB- Programms mit GUI zur Darstellung medizinischer Intensitätsbilder“ selbstständig, ohne unerlaubte fremde Hilfe und nur unter Verwendung der in der Arbeit aufgeführten Hilfsmittel angefertigt habe.

_____ _____

Ort, Datum Adrian Lühr

Anlagen

A MATLAB Skripte zu Kapitel 3

Anlage A1: MATLAB- Skript zur Histogrammspreizung

```
% Histogrammspreizung
clc, clear all

%% Einlesen des CT-Bildes
source_image = dicomread('CTScull_01.dcm');

%% Ausgabe des CT-Bildes ohne Wertebereichsanpassung
figure(1)
subplot(1,2,1)
imshow(source_image)                          % Darstellung des
Ausgangsbildes
title('CTScull\_01.dmc - Original');
colorbar

hist1=imhist(source_image);                   % Darstellung des
Histogramms
subplot(1,2,2)
bar(hist1)
xlim([0 255])
title('Histogramm - Ausgangssituation'); xlabel('Graustufen');

%% Ausgabe des CT-Bildes mit Wertebereichsanpassung
source_image_trans=double(source_image);      % casting nötig, um die
Berechnung durchzuführen
output_image=zeros(256,256);                  % Speicher reservieren für
das Zielbild

% Transformation der Grauwerte nach Formel (3.1)
for x=1:256
    for y=1:256
        output_image(x,y)=(intmax(class(source_image))-
intmin(class(source_image)))/...
                          (max(source_image_trans(:))-
min(source_image_trans(:)))*...
                          (source_image_trans(x,y)-
min(source_image_trans(:)))+intmin(class(source_image));
    end
end

output_image=int16(output_image);             % casting zum
Ausgangsdatentyp
figure(2)
subplot(1,2,1)
imshow(output_image);                         % Darstellung des
transformierten Bildes
title('CTScull\_01.dmc - Transformiert');
colorbar

hist2=imhist(output_image);                   % Darstellung des
Histogramms
```

```
subplot(1,2,2)
bar(hist2)
xlim([0 255])
title('Histogramm - Transformiertes Bild'); xlabel('Graustufen');

%% Ausgabe der Transformationskennlinie
g=-1024:1962;                                                    % X-
Achse
g_trans=(32767-(-32768))/(1962-(-1024))*(g-(-1024))+(-32768);    % Y-
Achse
figure(3)
plot(g,g_trans)                                  % Darstellung der
Transformationskennlinie
xlim([-32768 32767]); ylim([-32768 32767])
title('Kennlinie der linearen Histogrammspreizung');
xlabel('alter Grauwert'); ylabel('neuer Grauwert'); grid on;
```

Anlage A2: MATLAB- Skript zum Window- Leveling

```
% Window-Leveling
clc; clear all

%% Einlesen des CT-Bildes
ctimage = dicomread('CTScull_01.dcm');

%% Vektoren für width und level (nach erweiterter Aufgabenstellung)
width=[1500 2000 350 100];
level=[300 -200 50 35];

%% Bestimmung von gmin/gmax für die jeweiligen Fenster
for i=1:4
    gmin(i)=level(i)-width(i)/2;
    gmax(i)=level(i)+width(i)/2;
end

%% Ausgabe der transformierten Bilder
subplot(2,2,1)
imshow(ctimage,[gmin(1) gmax(1)])
title('Knochenfenster')

subplot(2,2,2)
imshow(ctimage,[gmin(2) gmax(2)])
title('Lungenfenster')

subplot(2,2,3)
imshow(ctimage,[gmin(3) gmax(3)])
title('Weichteilfenster')

subplot(2,2,4)
imshow(ctimage,[gmin(4) gmax(4)])
title('Gehirnfenster')
```

Anlage A3: MATLAB- Skript zur Histogrammäqualisation

```
%Histogrammäqualisation
clc; clear all;

%% Einlesen des MRT-Bildes
mrimage = dicomread('MRIScull_01.dcm');
mrimagecast=uint8(mrimage);

%% Darstellung des Quellbildes + Histogramm
figure(1)
subplot(1,2,1); imshow(mrimagecast)
title('MRIScull\_01.dcm - Original')
hist = imhist(mrimagecast);
subplot(1,2,2); bar(hist)
xlim([0 255])
title('Histogramm - Originalbild'); xlabel('Grauwerte');

%% Darstellung des äqualisierten Bildes + Histogramm
figure(2)
eq_mrimage=histeq(mrimagecast);
subplot(1,2,1); imshow(eq_mrimage)
title('MRIScull\_01.dcm - transformiert')
hist = imhist(eq_mrimage);
subplot(1,2,2); bar(hist)
xlim([0 255])
title('Histogramm - transformiertes Bild'); xlabel('Grauwerte');
```

Anlage A4: MATLAB- Skript zur Histogrammhyperbolisation

```
% Histogrammhyperbolisation
clc; clear all;

%% Einlesen des MRT-Bildes
mrimage = dicomread('MRIScull_01.dcm');
mrimagecast=uint8(mrimage);

a=-0.8; image=mrimagecast;  % Festlegen der Parameter

%% Kontrolle der Eingangsparameter
if strcmp(class(image),'uint8')& a>-1 & a<=0
    fprintf('\nParameter OK!\n')
else
    error('Fehler in Übergabeparameter! Bitte Prüfen!')
end

%% Histogrammhyperbolisation
LUT=zeros(3,256);                              % Lookup- Tabelle
[counts,x]=imhist(image);                      % Berechnung des
Histogramms
LUT(1,:)=x;                                    % Zeile 1: mögliche
Grauwerte
LUT(2,:)=counts;                               % Zeile 2: absolute
Häufigkeit der Grauwerte
```

```
sum_rel_freq=cumsum(LUT(2,:)./sum(counts));     % Summe der relativen
Häufigkeiten
LUT(3,:)=round(255*(sum_rel_freq).^(1/(a+1)));  % Zeile 3:
transformierte Grauwerte


%% Speicherreservierung für das Zielbild
img_output=uint8(zeros(256,256));

%% Transformation des Bildes
for x=1:256                                     % Iteration über das
gesamte Bild
    for y=1:256
        grauwert_ist=image(y,x);                % Auslesen des
Grauwertes
        grauwert_neu=LUT(3,grauwert_ist+1);     % Nachschlagen in der
LUT
        img_output(y,x)=grauwert_neu;           % transformierten
Grauwert schreiben
    end
end


%% Darstellung Quell- und Ausgabebild
subplot(1,2,1)
imshow(image); title('Quellbild');
subplot(1,2,2)
imshow(img_output); title('Transformiertes Bild');

%% Ausgabe der Transformationskennlinie
figure(2)
plot(LUT(1,:),LUT(3,:))
xlim([0 255]); ylim([0 255])
title('Kennlinie der Histogrammhyperbolisation');
xlabel('alter Grauwert'); ylabel('neuer Grauwert'); grid on;
```

B Quellcode der MATLAB- Applikation

Anlage B1: MATLAB- Funktion *grayValueTransformations_GUI.m*

```
function varargout = grayValueTransformations_GUI(varargin)
% GRAYVALUETRANSFORMATIONS_GUI MATLAB code for
grayValueTransformations_GUI.fig
%      GRAYVALUETRANSFORMATIONS_GUI, by itself, creates a new
GRAYVALUETRANSFORMATIONS_GUI or raises the existing
%      singleton*.
%
%      H = GRAYVALUETRANSFORMATIONS_GUI returns the handle to a new
GRAYVALUETRANSFORMATIONS_GUI or the handle to
%      the existing singleton*.
%
%
GRAYVALUETRANSFORMATIONS_GUI('CALLBACK',hObject,eventData,handles,...)
calls the local
%      function named CALLBACK in GRAYVALUETRANSFORMATIONS_GUI.M with
the given input arguments.
%
%      GRAYVALUETRANSFORMATIONS_GUI('Property','Value',...) creates a
new GRAYVALUETRANSFORMATIONS_GUI or raises the
%      existing singleton*.  Starting from the left, property value
pairs are
%      applied to the GUI before
grayValueTransformations_GUI_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property
application
%      stop.  All inputs are passed to
grayValueTransformations_GUI_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows
only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help
grayValueTransformations_GUI

% Last Modified by GUIDE v2.5 17-May-2013 13:49:34

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn',
@grayValueTransformations_GUI_OpeningFcn, ...
                   'gui_OutputFcn',
@grayValueTransformations_GUI_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
```

```
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before grayValueTransformations_GUI is made
visible.
function grayValueTransformations_GUI_OpeningFcn(hObject, eventdata,
handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to grayValueTransformations_GUI
(see VARARGIN)

% Choose default command line output for grayValueTransformations_GUI
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

axis off;           % Entfernen der Achsenbeschriftung

% Ändern der Button- Farbe
set(handles.load_button,'BackgroundColor','yellow');
set(handles.lineare_hist_button,'BackgroundColor','yellow');
set(handles.window_level_button,'BackgroundColor','yellow');
set(handles.histeq_button,'BackgroundColor','yellow');
set(handles.histhyp_button,'BackgroundColor','yellow');
set(handles.save_button,'BackgroundColor','yellow');
set(handles.print_button,'BackgroundColor','yellow');

guidata(hObject, handles);  % Update handles structure

% UIWAIT makes grayValueTransformations_GUI wait for user response
(see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = grayValueTransformations_GUI_OutputFcn(hObject,
eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in load_button.
function load_button_Callback(hObject, eventdata, handles)
% hObject    handle to load_button (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
```

```
% handles    structure with handles and user data (see GUIDATA)

%% Einlesen des CT-Bildes
handles.image_source=dicomread(uigetfile({'*.dcm','DICOM- file
(*.dcm)'},'Pick a DICOM- file'));
imshow(handles.image_source)

handles.alpha=0;    % Alpha inizialisieren und Wert zuweisen
set(handles.alpha_popup,'Value',1)

% Ändern der Button- Farbe
set(handles.load_button,'BackgroundColor','green');
set(handles.save_button,'BackgroundColor','green');
set(handles.print_button,'BackgroundColor','green');
if      1 == get(handles.ct_radiobutton,'Value')

set(handles.lineare_hist_button,'BackgroundColor','green');

set(handles.window_level_button,'BackgroundColor','green');
            set(handles.histeq_button,'BackgroundColor','yellow');
            set(handles.histhyp_button,'BackgroundColor','yellow');
elseif  1 == get(handles.mr_radiobutton,'Value')

set(handles.lineare_hist_button,'BackgroundColor','green');

set(handles.window_level_button,'BackgroundColor','yellow');
            set(handles.histeq_button,'BackgroundColor','green');
            set(handles.histhyp_button,'BackgroundColor','green');
end
guidata(hObject, handles);


% --------------------------------------------------------------------
function file_menu_Callback(hObject, eventdata, handles)
% hObject    handle to file_menu (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --------------------------------------------------------------------
function file_menu_exit_Callback(hObject, eventdata, handles)
% hObject    handle to file_menu_exit (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
delete(handles.figure1);


% --- Executes on button press in lineare_hist_button.
function lineare_hist_button_Callback(hObject, eventdata, handles)
% hObject    handle to lineare_hist_button (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
hist_stretch(handles.image_source)

% --- Executes on button press in window_level_button.
function window_level_button_Callback(hObject, eventdata, handles)
% hObject    handle to window_level_button (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
if      1 == get(handles.mr_radiobutton,'Value')
    warndlg('Das MRT- Bild ist für das Window- Leveling nicht
geeignet!')
end
if 1 == get(handles.knochenfenster_radiobutton,'Value')
    window_level(handles.image_source,1)
elseif 1 == get(handles.lungenfenster_radiobutton,'Value')
    window_level(handles.image_source,2)
elseif 1 == get(handles.weichteilfenster_radiobutton,'Value')
    window_level(handles.image_source,3)
elseif 1 == get(handles.gehirnfenster_radiobutton,'Value')
    window_level(handles.image_source,4)
end



% --- Executes on button press in histeq_button.
function histeq_button_Callback(hObject, eventdata, handles)
% hObject    handle to histeq_button (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
if      1 == get(handles.ct_radiobutton,'Value')
    warndlg('Das CT- Bild ist für die Histogrammäqualisation nicht
geeignet!')
end
hist_eq(handles.image_source);

% --- Executes on button press in histhyp_button.
function histhyp_button_Callback(hObject, eventdata, handles)
% hObject    handle to histhyp_button (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
if      1 == get(handles.ct_radiobutton,'Value')
    warndlg('Das CT- Bild ist für die Histogrammhyperbolisation nicht
geeignet!')
end
imshow(hist_hyp(handles.image_source,handles.alpha));


% --------------------------------------------------------------------
function help_menu_Callback(hObject, eventdata, handles)
% hObject    handle to help_menu (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --------------------------------------------------------------------
function help_menu_info_Callback(hObject, eventdata, handles)
% hObject    handle to help_menu_info (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
grayValue_Info_GUI

% --------------------------------------------------------------------
function help_menu_help_Callback(hObject, eventdata, handles)
% hObject    handle to help_menu_help (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
grayValue_Help_GUI


% --- Executes on button press in save_button.
function save_button_Callback(hObject, eventdata, handles)
% hObject    handle to save_button (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
F=getframe(handles.axes1);          %select axes in GUI
figure();                           %new figure
image(F.cdata);                     %show selected axes in new figure
title('Created with GrayValueTransformations')
axis equal; axis off;               %delete axis
print(gcf, '-dpng', uiputfile({'*.png','Image File (*.png)'},'Save
as'));  %save figure
close(gcf);                         %close figure


% --- Executes on button press in print_button.
function print_button_Callback(hObject, eventdata, handles)
% hObject    handle to print_button (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
F=getframe(handles.axes1);          %select axes in GUI
figure(1);                           %new figure
image(F.cdata);                     %show selected axes in new figure
title('Created with GrayValueTransformations')
axis equal; axis off;               %delete axis
printdlg(figure(1))                 %print figure
close(gcf);                         %close figure


% --- Executes on selection change in alpha_popup.
function alpha_popup_Callback(hObject, eventdata, handles)
% hObject    handle to alpha_popup (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

val=get(hObject,'Value');        %Nummer des Menü- Eintrags
str=get(hObject,'String');       %Liste der Einträge im Popup- Menü

switch(str{val})                 % Auswahl des Eintrags zur Nummer
    case 'alpha= 0'
        handles.alpha=0;
    case 'alpha= -1/3'
        handles.alpha=-1/3;
    case 'alpha= -0.5'
        handles.alpha=-0.5;
    case 'alpha= -2/3'
        handles.alpha=-2/3;
    case 'alpha= -0.8'
        handles.alpha=-0.8;
end

guidata(hObject,handles)

% --- Executes during object creation, after setting all properties.
function alpha_popup_CreateFcn(hObject, eventdata, handles)
% hObject    handle to alpha_popup (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

Anlage B2: MATLAB- Funktion *hist_stretch.m*

```
function hist_stretch(image)
% Funktion zur linearen Histogrammspreizung
% Übergabeparameter: Das zu Transformierende Bild

imshow(image,[min(image(:)) max(image(:))])
```

Anlage B3: MATLAB- Funktion *window_level.m*

```
function image_output=window_level(image_source,window_id)
% Funktion zur Durchführung des Window-Leveling
% Parameter(Quellbild, ID des Fensters)


%% Vektoren für width und level
width=[1500 2000 350 100];
level=[300 -200 50 35];

%% Bestimmung von gmin/gmax für das gewünschte Fenster
gmin=level(window_id)-width(window_id)/2;
gmax=level(window_id)+width(window_id)/2;


%% Ausgabe des transformierten Bildes
imshow(image_source,[gmin gmax])
```

Anlage B4: MATLAB- Funktion *hist_eq.m*

```
function image_output=hist_eq(image_source)
%Histogrammäqualisation
%Übergabeparameter: Quellbild

%% Casting des Bildes
image=uint8(image_source);

%% Histogrammäqualisation und Ausgabe
image_output=histeq(image);
imshow(image_output)
```

Anlage B5: MATLAB- Funktion *hist_hyp.m*

```
function img_output=hist_hyp(image_source,a)
% Funktion zur Transformation von uint8 Grauwertbildern
% Übergabeparameter: image= Quellbild; a=alpha

image=uint8(image_source);

%% Kontrolle der Eingangsparameter
if strcmp(class(image),'uint8')& a>=-1 & a<=0
    fprintf('\nParameter OK!\n')
else
    error('Fehler in Übergabeparameter! Bitte Prüfen!')
end

%% Histogrammhyperbolisation
LUT=zeros(3,256);                                   % Lookup- Tabelle
[counts,x]=imhist(image);                           % Berechnung des
Histogramms
LUT(1,:)=x;                                         % Zeile 1: mögliche
Grauwerte
LUT(2,:)=counts;                                    % Zeile 2: absolute
Häufigkeit der Grauwerte
sum_rel_freq=cumsum(LUT(2,:)./sum(counts));         % Summe der relativen
Häufigkeiten
LUT(3,:)=round(255*(sum_rel_freq).^(1/(a+1)));      % Zeile 3:
transformierte Grauwerte


%% Speicherreservierung für das Zielbild
img_output=uint8(zeros(256,256));

%% Transformation des Bildes
for x=1:256                                         % Iteration über das
gesamte Bild
    for y=1:256
        grauwert_ist=image(y,x);                    % Auslesen des
Grauwertes
        grauwert_neu=LUT(3,grauwert_ist+1);         % Nachschlagen in der
LUT
        img_output(y,x)=grauwert_neu;               % transformierten
Grauwert schreiben
    end
end
```

Anlage B6: MATLAB- Funktion *grayValue_Info.GUI.m*

```
function varargout = grayValue_Info_GUI(varargin)
% GRAYVALUE_INFO_GUI MATLAB code for grayValue_Info_GUI.fig
%      GRAYVALUE_INFO_GUI, by itself, creates a new GRAYVALUE_INFO_GUI
or raises the existing
%      singleton*.
%
%      H = GRAYVALUE_INFO_GUI returns the handle to a new
GRAYVALUE_INFO_GUI or the handle to
%      the existing singleton*.
%
```

```
%      GRAYVALUE_INFO_GUI('CALLBACK',hObject,eventData,handles,...)
calls the local
%      function named CALLBACK in GRAYVALUE_INFO_GUI.M with the given
input arguments.
%
%      GRAYVALUE_INFO_GUI('Property','Value',...) creates a new
GRAYVALUE_INFO_GUI or raises the
%      existing singleton*.  Starting from the left, property value
pairs are
%      applied to the GUI before grayValue_Info_GUI_OpeningFcn gets
called.  An
%      unrecognized property name or invalid value makes property
application
%      stop.  All inputs are passed to grayValue_Info_GUI_OpeningFcn
via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows
only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help
grayValue_Info_GUI

% Last Modified by GUIDE v2.5 11-May-2013 16:07:08

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @grayValue_Info_GUI_OpeningFcn,
...
                   'gui_OutputFcn',  @grayValue_Info_GUI_OutputFcn,
...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before grayValue_Info_GUI is made visible.
function grayValue_Info_GUI_OpeningFcn(hObject, eventdata, handles,
varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to grayValue_Info_GUI (see
VARARGIN)
```

```
% Choose default command line output for grayValue_Info_GUI
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes grayValue_Info_GUI wait for user response (see
UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = grayValue_Info_GUI_OutputFcn(hObject, eventdata,
handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;
```

Anlage B7: MATLAB- Funktion *grayValue_Help_GUI.m*

```
function varargout = grayValue_Help_GUI(varargin)
% GRAYVALUE_HELP_GUI MATLAB code for grayValue_Help_GUI.fig
%      GRAYVALUE_HELP_GUI, by itself, creates a new GRAYVALUE_HELP_GUI
or raises the existing
%      singleton*.
%
%      H = GRAYVALUE_HELP_GUI returns the handle to a new
GRAYVALUE_HELP_GUI or the handle to
%      the existing singleton*.
%
%      GRAYVALUE_HELP_GUI('CALLBACK',hObject,eventData,handles,...)
calls the local
%      function named CALLBACK in GRAYVALUE_HELP_GUI.M with the given
input arguments.
%
%      GRAYVALUE_HELP_GUI('Property','Value',...) creates a new
GRAYVALUE_HELP_GUI or raises the
%      existing singleton*.  Starting from the left, property value
pairs are
%      applied to the GUI before grayValue_Help_GUI_OpeningFcn gets
called.  An
%      unrecognized property name or invalid value makes property
application
%      stop.  All inputs are passed to grayValue_Help_GUI_OpeningFcn
via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows
only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES
```

```
% Edit the above text to modify the response to help
grayValue_Help_GUI

% Last Modified by GUIDE v2.5 11-May-2013 16:07:30

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @grayValue_Help_GUI_OpeningFcn,
...
                   'gui_OutputFcn',  @grayValue_Help_GUI_OutputFcn,
...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before grayValue_Help_GUI is made visible.
function grayValue_Help_GUI_OpeningFcn(hObject, eventdata, handles,
varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to grayValue_Help_GUI (see
VARARGIN)

% Choose default command line output for grayValue_Help_GUI
handles.output = hObject;
% Update handles structure
guidata(hObject, handles);

% UIWAIT makes grayValue_Help_GUI wait for user response (see
UIRESUME)
% uiwait(handles.figure1);

% --- Outputs from this function are returned to the command line.
function varargout = grayValue_Help_GUI_OutputFcn(hObject, eventdata,
handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;
```

C Übersicht der auf dem Datenträger vorhandenen Dateien

Auf dem beigelegten Datenträger befinden sich alle Dateien, die im Zusammenhang mit dieser Arbeit erstellt worden sind. Zusätzlich ist diese Übersicht auf der beigelegten DVD unter „readme.txt“ gespeichert.

Ordner: Ausarbeitung:

- Ausarbeitung.docx
- Ausarbeitung.pdf

Ordner: MATLAB_Skripte

- Histogrammspreizung.m
- window_leveling.m
- Histogrammausgleich.m
- Histogrammhyperbolisation.m
- CTScull_01.dcm
- MRIScull_01.dcm

Ordner: Applikation

- grayValueTransformations_GUI.m
- grayValueTransformations_GUI.fig
- grayValue_Info_GUI.m
- grayValue_Info_GUI.fig
- grayValue_Help_GUI.m
- grayValue_Help_GUI.fig
- hist_stretch.m
- window_level.m
- hist_eq.m
- hist_hyp.m
- CTScull_01.dcm
- MRIScull_01.dcm