Entwicklung eines Verfahrens zur semi-automatisierten Konfiguration und Analyse von Bildsegmentierungsalgorithmen

Abschlussarbeit zur Erlangung des akademischen Grades
Master of Engineering (M.Eng.)

vorgelegt von
Georg Kuhne

Referent: Prof. Dr. Michael Schenke
Koreferent: Florian Gehrke [Dipl. -Bioinf.]

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst und keine anderen Hilfsmittel als angegeben verwendet habe. Insbesondere versichere ich, dass ich alle wörtlichen und sinngemäßen Übernahmen aus anderen Werken als solche kenntlich gemacht habe.

Halle, den 30.04.2016

Abstract(deutsch)

Ein Teilgebiet der Bildverarbeitung ist die Bildsegmentierung. Die Bildsegmentierung umfasst eine Vielzahl von Verfahren um Bilder anhand von Homogenitätskriterien in sinnvolle Teilbereiche einzuteilen. In dieser Arbeit wird ein Verfahren vorgestellt, mit dem sich die Verfahren der Bildsegmentierung semiautomatisch konfigurieren und auf ihre Eignung für das jeweilige zu segmentierende Bild überprüfen lassen.

Abstract(English)

One of the topics of image processing is image segmentation. The goal of image segmentation is to partition a digital image into multiple segments by specific criteria. Therefore, there are many algorithms. In this master thesis a method is presented, which allows to configure and analyze segmentation algorithms semi automatically.

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation

Die Problemstellung dieser Arbeit begegnete dem Autor bei seiner Tätigkeit als Software-Entwickler bei dem Unternehmen *Kapelan Bio-Imaging* (Kapelan). Kapelan wurde im Jahre 2000 gegründet und entwickelt Software- und Hardwarelösungen zur Unterstützung und Automatisierung von Auswertungsprozessen in Medizin, Biologie und Forschung. Das Unternehmen ist auf Testverfahren spezialisiert, in denen die Auswertung auf Basis visuell wahrnehmbarer Eigenschaften von Proben erfolgt. (vgl. [KAP16b]) So bietet Kapelan einerseits Hardware-Produkte zur Bildaufnahme an wie den *LabImager TR* – ein Gerät zur Aufnahme von sogenannten Immo-Strips, der in Abbildung 1 zu sehen ist. (siehe [KAP16c])

Abbildung 1: LabImager TR (Quelle: [KAP16c])

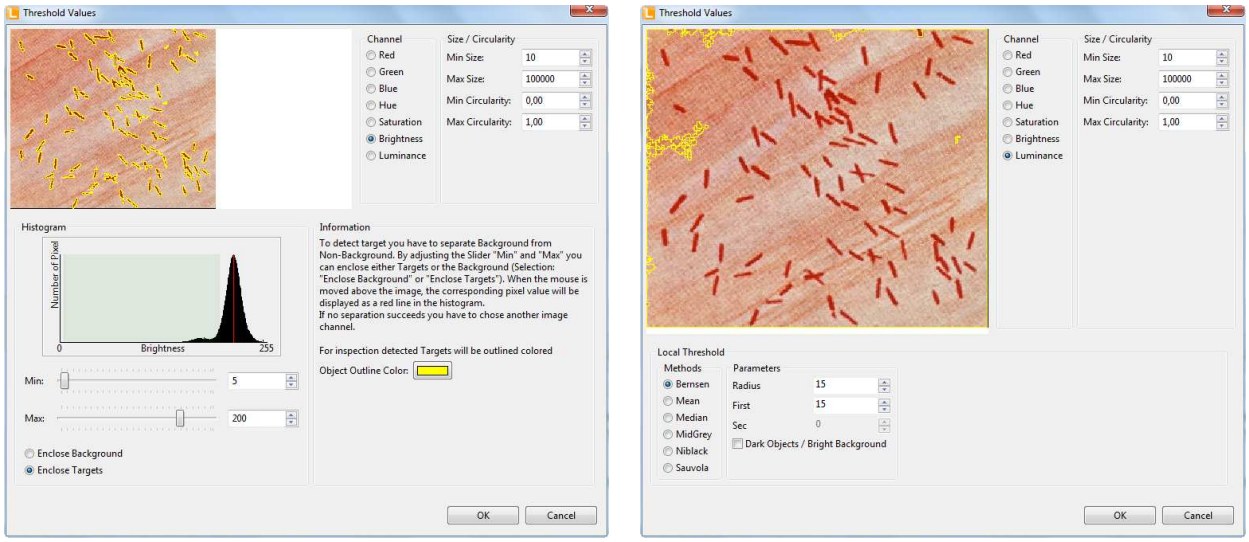
Andererseits bietet LabImage auch eine Reihe von Software-Produkten zur Analyse und Auswertung von bereits vorliegendem digitalen Bildmaterial an. Ein Beispiel hierfür ist LabImage CC – eine Anwendung zum Detektieren und Zählen von Bakterien und Zellenkolonien. (siehe [KAP16a])

Das Bearbeiten und Analysieren von digitalen Bildern sprich die Bildverarbeitung ist ein Kernelement in der von Kapelan entwickelten Software. Ein Teilgebiet davon, welches grundlegend für viele Anwendungen Kapelans ist, ist die Bildsegmentierung (Segmentierung). Unter Segmentierung versteht man das Erkennen und Bestimmen inhaltlich zusammengehöriger Regionen in einem Bild.[1] In der Regel kann ein Mensch diese Aufgabe leicht und intuitiv lösen. So dürfte es beispielsweise für den Leser dieser Arbeit kein Problem darstellen die weißen Kolonien[2] in Abbildung 2 zu identifizieren. Soll dies nun rechnergestützt oder sogar automatisiert erfolgen, so bedarf es der Auswahl, Konfiguration und Anwendung eines geeigneten Segmentierungsalgorithmus. Die Voraussetzung dafür sind Kenntnisse über die Funktionsweisen der Segmentierungsalgorithmen und das Wissen um deren Eignung für das vorliegende Bildmaterial. Dies ist keineswegs trivial.

[1] Ausführlicher wird sich mit der Bildsegmentierung in Abschnitt 3 auseinandergesetzt.
[2] Auf biowissenschaftliche Aspekte der Beispielbilder soll in dieser Arbeit nicht eingegangen werden.

Abbildung 2: Beispielbild für LabImage CC (Quelle: [CCE16])

Kapelan bietet beispielsweise in seiner Anwendung *LabImage CC* zwei Segmentierungsverfahren an, ein lokales und globales Schwellwertverfahren. Einen Überblick über die Konfigurationsmöglichkeiten dieser Verfahren gibt Abbildung 3. Aus ihr wird ersichtlich, dass für die Anwendung der Verfahren jeweils zahlreiche Einstellungen vom Benutzer vorgenommen werden können. So muss der Benutzer beim globalen Schwellwertverfahren beispielsweise unter anderem den Farbkanal (Channel), ein Minimum und Maximum des Schwellwertes auswählen. Die genaue Bedeutung dieser Parameter, wird im Abschnitt 3 dieser Arbeit behandelt. An dieser Stelle soll lediglich verdeutlicht werden, dass die Software an fortgeschrittene Benutzer adressiert ist, denn sie verlangt dem Benutzer bei der Bedienung ein enormes Hintergrundwissen über Bildverarbeitung ab.



globales Schwellwertverfahren lokales Schwellwertverfahren

Abbildung 3: Screenshots der Dialoge zur Konfiguration der lokalen und globalen Schwellwertverfahren

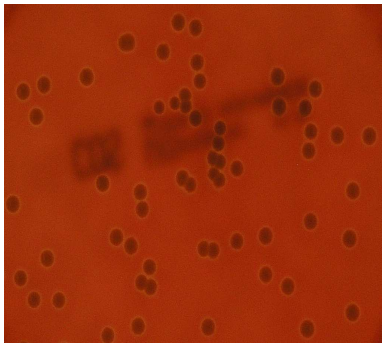
Die Bildsegmentierung auch für auch für weniger versierte Anwender zu ermöglichen ist das grundlegende Motiv dieser Arbeit.

1.2 Zielstellung, Anwendungsfälle und Anforderung

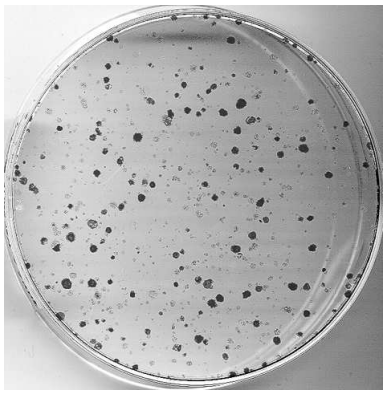
Ziel dieser Arbeit ist es, ein mächtiges und ergonomisches Verfahren für die Bildsegmentierung zu entwickeln.

Wenn man sich einen aus Sicht des Benutzers idealen rechnergestützten Bildsegmentierungsvorgang vorstellt, so sollte dieser vollautomatisch ablaufen und universell – also für jedes Bild – einsetzbar sein. Darüber hinaus sollte das mit ihm erreichte Ergebnis korrekt sein, sprich die für den Menschen homogen erscheinenden Bereiche des Bildes sollten durch ihn gefunden werden.

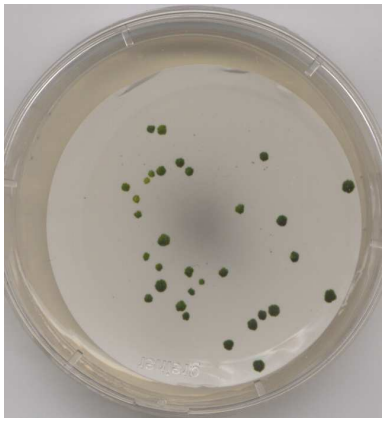
Die Entwicklung eines derartigen Verfahrens, erscheint angesichts der Fülle an verschiedenen Bildtypen unmöglich. Abbildung 4 zeigt eine kleine Auswahl möglicher Bilder, die mit *LabImage CC* bearbeitet werden sollen um den Variantenreichtum von zu analysierenden Bildern zu verdeutlichen. Es ist nicht absehbar, welche Eigenschaften ein zu analysierendes Bild hat.



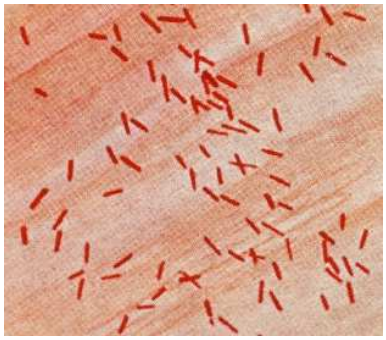
(a) Störungen im Hintergrund, rote Färbung



(b) unterschiedliche Größe und Sättigung der Kolonien, Kolonien am Rand



(c) inhomogener HG, Schatten, Spiegelungen



(d) inhomogener HG, keine runde Form, Stäbchen

Abbildung 4: Weitere Beispielbilder für LabImage CC

Aus diesem Grund erscheint es sinnvoll, die Anforderungen an ein ideales Bildsegmentierungsverfahrens einzuschränken. Dies kann auf zwei Arten geschehen. Entweder durch die Aufgabe des universellen Charakters in der Form, dass nur bestimmte Bildtypen bearbeitet werden können, oder durch die Aufgabe des Prinzip des voll-automatischen Ablaufs des Verfahrens. Erstere widerspricht dem Umstand des eben angedeuteten Variantenreichtums der mit *LabImage CC* zu bearbeitenden Bilder. Deswegen ist der Autor dieser Arbeit, zu der Einsicht gelangt, statt eines Konzepts für vollautomatische Segmentierung, ein Konzept für semiautomatische Segmentierung zu entwickeln. Die Segmentierung soll jedoch in ihrer Bedienung intuitiv gestaltet so werden, dass Seitens des Benutzers möglichst keine Kenntnisse der Bildverarbeitung notwendig sein sollen.

Abbildung 5 gibt einen Überblick über die Anwendungsfälle, denen das zu entwickelnde Verfahren gerecht werden muss, und aus denen sich weitere Anforderungen ableiten lassen.

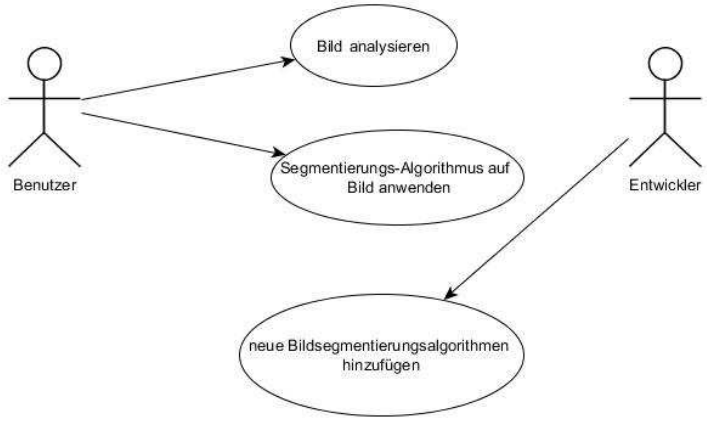


Abbildung 5: Anwendungsfalldiagramm des zu entwickelnden Verfahrens

Das zu entwickelnde Verfahren lässt sich aus der Sicht zweier Akteure betrachten, einerseits aus Sicht des Endanwenders oder Benutzers der Software und aus Sicht des Software-Entwicklers. Der erste Anwendungsfall des Benutzers ist das Analysieren eines vorliegenden Bildes. In diesen Vorgang soll einerseits ein geeigneter Algorithmus für die Segmentierung des Bildes ermittelt und semiautomatisch konfiguriert werden. Der zweite Anwendungsfall des Benutzers ist das Anwenden des gefundenen Algorithmus auf einen Bild. Dies kann das Bild sein, dass bereits analysiert worden ist. Häufig müssen mehrere Bilder eines Typs, die Beispielsweise in einer biowissenschaftlichen Testreihe entstanden sind, bearbeitet werden. Hierbei ist es naheliegend, dass die Auswahl und Konfiguration des Segmentierungsalgorithmus anhand eines dieser Bilder erfolgen soll und dann auf alle anderen Bilder des gleichen Typs angewendet werden. Eine bereits

erstellte Analyse [1] soll also wiederverwendbar sein. Aus dem Umstand, dass das Analysieren eines Bildes nur einmalig für eine Klasse von Bildern und das Anwenden einer Analyse auf jedes zu bearbeitende Bild erfolgen muss, lassen sich Anforderungen bezüglich des Rechenaufwands schlussfolgern. So ist es sinnvoll, den Rechen- und Zeitaufwand für das Anwenden einer Analyse möglichst gering zu halten, während bei der Erstellung der Analyse diesbezüglich nur wenig Rücksicht genommen werden muss.

Der Software-Entwickler hingegen hat nur einen Anwendungsfall: das Verfahren um neue Algorithmen zu erweitern. Daraus resultiert die Anforderung, dass das Verfahren mit geringem Anpassungsaufwand für beliebige Segmentierungsalgorithmen erweiterbar sein muss.

1.3 Aufbau der Arbeit

Der weitere Aufbau dieser Arbeit gestaltet sich wie folgt: Zunächst werden im Abschnitt 2 einige grundlegende Begriffe und Zusammenhänge der Bildverarbeitung vorgestellt. Im darauffolgenden Abschnitt 3 werden die Problemstellung der Bildsegmentierung sowie einige Lösungsansätze und Verfahren verdeutlicht. Im Abschnitt 4 erfolgt die Erläuterung des entwickelten Verfahrens zur Bildsegmentierung. Abschnitt 4 verdeutlicht die Umsetzbarkeit des Verfahrens anhand einer prototypischen Implementierung. Abschließend erfolgt im Abschnitt 6 eine Bewertung und das Aufzeigen möglicher Erweiterungen des Verfahrens .

[1] Ein ausgewählter Algorithmus mit seiner ermittelten Konfiguration soll im Kontext des zu entwickelnden Verfahrens Analyse genannt werden.

2 Grundlagen der Bildverarbeitung

2.1 Das digitale Bild als diskrete Ortsbereichsfunktion

Ein Bild lässt sich als eine Funktion f auffassen, die einen Definitionsbereich D in einem Farbraum F abbildet :

$$f : \mathbb{D} \to F \tag{1}$$

Mit dieser Definition können Bilder im Allgemeinen formal beschrieben werden. Geht man von einem Idealbild aus, so hat f die Eigenschaft, dass sowohl D und F kontinuierlich sind. Der Definitionsbereich D ist dabei gegeben durch Punkte der Gestalt $(x, y) \in \mathbb{R} \times \mathbb{R}$. Jedem dieser Punkte wird mit f ein Farbwert aus F zugeordnet. Je nach Farbraum kann F mehrere Dimensionen haben. Die Elemente von F können im idealen Bild demnach eine reelle Zahl oder ein Tupel von reellen Zahlen sein.

Möchte man digitale Bilder mit einem solchen Model darstellen, so unterscheidet sich das Model von dem eines idealen Bildes darin, dass bei ihnen sowohl ihr Definitionsbereich als auch ihr Farbraum (Wertebereich) diskret sind. (vgl. [TL97] S. 99) Die Elemente von D haben demnach die Gestalt $(x, y) \in \mathbb{N} \times \mathbb{N}$, während die Elemente von F natürliche Zahlen, oder Tupel von natürlichen Zahlen sind. Bei der Transformation eines idealen Bildes in ein digitales Bild sind der Definitionsbereich durch Abtastung und der Wertebereich durch Quantisierung zu diskretisieren. Dies wird beispielsweise in einem Bild-Scanner technisch realisiert.

Das eben vorgestellte Model, kann als eine formale Beschreibung von Pixelgrafiken bzw. Rastergrafiken aufgefasst werden. Pixelgrafiken sind eine computerlesbare Form von Bildern, in denen in einem Rechteck angeordneten Pixeln (Bildpunkten) jeweils ein Farbwert zugeordnet wird. (vgl. [WIK16i]) Die Breite und Höhe des Bildes oder des Rechteckes entspricht dabei der Anzahl der Pixel, die vertikal beziehungsweise horizontal angeordnet sind.

Neben Pixelgrafiken sind auch andere Darstellungsformate für digitale Bilder für die Computergrafik bedeutsam. Ein Beispiel hierfür sind die Vektorgrafiken, in denen Bildinhalte mittels graphischer Primitiven wie Linien, Kreisen, Polygonen dargestellt werden. (vgl. [WIK16l]) Diese sind für diese Arbeit aber nicht relevant und es soll an dieser Stelle nur auf sie hingewiesen werden.

In diesem Abschnitt wurde zunächst die für diese Arbeit bedeutsame Form der Darstellung von digitalen Bildern vorgestellt. Im nächsten Abschnitt 2.2 sollen nun einige der für die Computergrafik wichtigen Farbräume erläutert werden.

2.2 Farbräume

2.2.1 RGB-Farbraum

Der RGB-Farbraum ist ein additiver Farbraum, der Farben durch das additive Mischen dreier Grundfarben (Rot, Grün und Blau) nachbildet. Er basiert auf der Dreifarbentheorie, einer historischen Theorie zur Farbwahrnehmung im menschlichen Auge, die 1850 von Hermann von Helmholtz entwickelt wurde. Helmholtz vermutete, dass es im menschlichen Auge drei Typen von Rezeptoren gibt, die unterschiedlich empfindlich auf Licht verschiedener Wellenlängen reagieren. (vgl. [WIK16a]) Der RGB-Farbraum findet als Funktionsprinzip von Farbfernsehbildschirmen und Farbmonitoren Anwendung. Abbildung 6 stellt den RGB-Farbraum als Farbwürfel dar. Die Punkte innerhalb des Würfels, werden mittels Werten für die Grundfarben Rot, Grün und Blau definiert und entsprechen den gemischten Farben, die durch die Farbanteile der Grundfarben definiert sind.

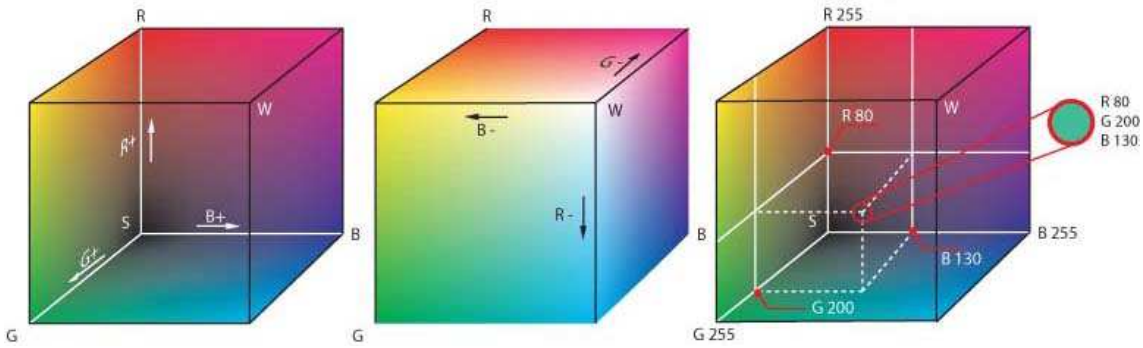


Abbildung 6: Darstellung des GBR-Farbraums als Farbwürfe (Quelle: [WIK16j])

In der Computergraphik ist es üblich, dass der RGB-Farbraum diskretisiert wird. Die möglichen Werte, die die Farbanteile der Grundfarben annehmen können, sind Ganzzahlen zwischen 0 und einer Maximalzahl. Die Maximalzahl ist dabei abhängig davon, wie fein oder grob, der RGB-Farbraum diskretisiert wurde. Übliche Maximalzahlen sind 7, 31, 255, 1023, 4095, 16383, 65535. (vgl. [WIK16j]) In dieser Arbeit wird bei Farbbildern eine Farbtiefe von 24-Bit angenommen. Das bedeutet, für jede Grundfarbe stehen 8 Bit zur Verfügung, mit denen sich ein Wertebereich von 0 bis 255 darstellen lässt.

2.2.2 HSV-Farbraum

Auch im HSV-Farbraum werden Farbeindrücke durch das Kombinieren dreier Grundgrößen beschrieben. Diese sind Farbton (engl. hue), Farbsättigung (engl. saturation) und Hellwert (oder Dunkelstufe) (engl. value). Der HSV-Farbraum bietet gegenüber dem RGB-Farbraum einen wesentlich intuitiveren Ansatz für die manuelle Definition einer Farbe. Eine Farbmischung kann dadurch erfolgen, dass man zunächst einen Farbton wählt und dann entscheidet, wie gesättigt und wie hell (oder dunkel) die gewünschte Farbe sein soll. (vgl. [WIK16d])

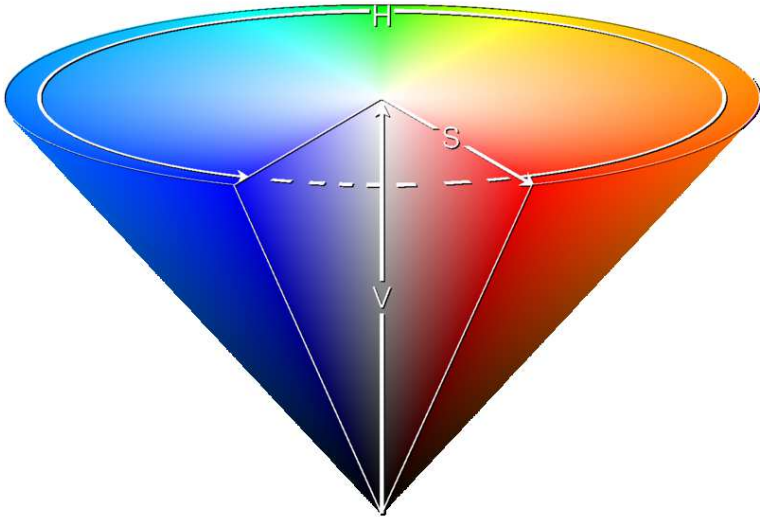


Abbildung 7: Darstellung des HSV-Farbraums als Kegel (Quelle: [WIK16d])

Ein Modell, das die HSV-Parameter anschaulich darstellt, ist der in Abbildung 7 dargestellte auf der Spitze stehende HSV-Kegel. Hierbei nimmt die Helligkeit (V) von unten nach oben zu. Der Farbton (H) wird durch einen Farbwinkel definiert, der Werte zwischen 0° und 360° annehmen kann. Die Sättigung (H) spezifiziert die Zumischung von purem Weiß, wobei mit zunehmender Sättigung eine Abnahme des Weißanteils bedeutet.

Analog zum RGB-Farbraum ist es für die digitale Repräsentation der Farbe notwendig die HSV-Parameter zu diskretisieren und zu beschränken. Soll ein Farbwert ebenfalls mit 24-Bit kodiert werden, so kann jeder HSV-Parameter Werte zwischen 0 und 255 annehmen.

2.2.3 Grauwert-Farbraum

Der Grauwert-Farbraum hat im Gegensatz zum RGB-Farbraum und HSV-Farbraum nur eine Dimension. Seine Werte repräsentieren Helligkeits-Stufen. Bei digitalen Bildern werden üblicherweise 8-Bit für den kodierten Grauwert verwendet. Damit lässt sich ein diskreter Wertebereich darstellen, der Werte von 0 für schwarz und 255 für weiß umfasst. Ein Bild, dessen Farbraum der Grauwert-Farbraum ist, wird auch Grauwertbild genannt.

2.2.4 Binärer-Farbraum

Der Binäre-Farbraum umfasst nur zwei Werte. Jeder Pixel kann also mit einem Bit gespeichert werden. Die Kodierung der Pixel erfolgt meist mit dem Wert 0 für Schwarz und 1 für Weiß. Ein Bild, dessen Farbraum der Binär-Farbraum ist, wird auch Binärbild genannt.

2.2.5 Transformationen zwischen den Farbräumen

Die Darstellung einer Farbe im RGB-Farbraum kann in eine Darstellung im HSV-Farbraum umgewandelt werden und umgekehrt. Die dafür notwendigen Berechnungen folgen dem Formelsatz von Gonzalez und Woods. (siehe [RG02] S. 295)

Für die Umrechnung von RGB nach HSV müssen die Farbwerte Rot (R), Grün (G) und Blau (B) so normiert sein, dass sie im Intervall [0,1] liegen. Es muss also gelten:

$$R, G, B \in [0, 1]$$

Die Normierung erfolgt, in dem man den einen Farbwert einer Grundfarbe durch den maximalen Wert, den diese Farbe annehmen kann, dividiert. Bei einer Farbtiefe von 24-Bit ist dieser Maximalwert beispielsweise 255.

Liegen die Werte von R, G und B in normierter Form vor, wird aus ihnen das Maximum (MAX) und das Minimum (MIN) bestimmt.

$$MAX := max(R, G, B), MIN := min(R, G, B)$$

Dann werden die Parameter des HSV-Farbraums H, S, V wie folgt bestimmt:

$$H := \begin{cases} 0°, & falls\, MAX = MIN \Leftrightarrow R = G = B \\ 60° \left(0 + \frac{G-B}{MAX-MIN}\right), & fals\, MAX = R \\ 60° \left(2 + \frac{B-R}{MAX-MIN}\right), & falls\, MAX = G \\ 60° \left(4 + \frac{R-G}{MAX-MIN}\right), & falls\, MAX = B \end{cases}$$

$$falls\, H < 0°\, dann H := H + 360°$$

$$S := \begin{cases} 0, & falls\, MAX = 0 \Leftrightarrow R = G = B = 0 \\ \frac{MAX-MIN}{MAX}, & sonst \end{cases}$$

$$V := MAX$$

Als Ergebnis der eben aufgeführten Berechnungen liegen H, S und V in folgender Form vor:

$H \in [0°, 360°]$und $S, V \in [0, 1]$

Bei Bedarf können die somit berechneten Werte wieder auf einen diskreten Wertebereich von 0-255 (24-Bit) abgebildet werden.

Für die Umrechnung von HSV-Farbwerten in RGB-Farbwerte müssen ebenfalls Vorbedingungen erfüllt werden. So muss gelten:

$$H \in [0°, 360°] \, und \, S, V \in [0, 1]$$

Zunächst berechnet man zwei Hilfswerte h und f:

$$h := \lfloor \tfrac{H}{60°} \rfloor \text{ [1]} \; ; \quad f := \left(\tfrac{H}{60°} - h\right)$$

Dann bestimme man p, q, t wie folgt:

$$p := V * (1 - S); \; q := V * (1 - S * f); \; t := V * (1 - S * (1 - f))$$

Nun können R, G und B wie folgt bestimmt werden:

$$(R, G, B) := \begin{cases} (V, t, p), & falls \, h \in \{0, 6\} \\ (q, V, p), & falls \, h = 1 \\ (p, V, t), & falls \, h = 2 \\ (p, q, V), & falls \, h = 3 \\ (t, p, V), & falls \, h = 4 \\ (V, p, q), & falls \, h = 5 \end{cases}$$

Als Ergebnis liegen die Werte R, G und B in normierter Form vor und können bei Bedarf auf andere Wertebereiche abgebildet werden.

$$R, G, B \in [0, 1]$$

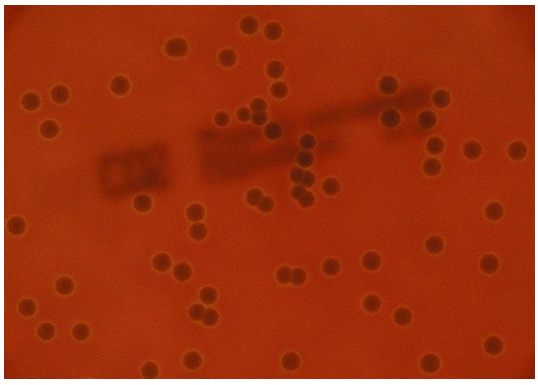
In den bisherigen Ausführungen dieses Abschnitts wurde deutlich, dass man eine RGB-Darstellung einer Farbe in eine äquivalente HSV-Darstellung überführen kann und umgekehrt. Daraus lässt sich schlussfolgern, dass sich auch ein Bild, dessen Farben mit dem RGB-Farbraum dargestellt sind (RGB-Bild), in ein äquivalentes Bild, dessen Farben im HSV-Farbraum dargestellt sind (HSV-Bild), transformieren lässt. Dies kann erfolgen, indem man die jeweilige Farbtransformation für jeden Pixel des Bildes anwendet.

2.2.6 Farbkanäle

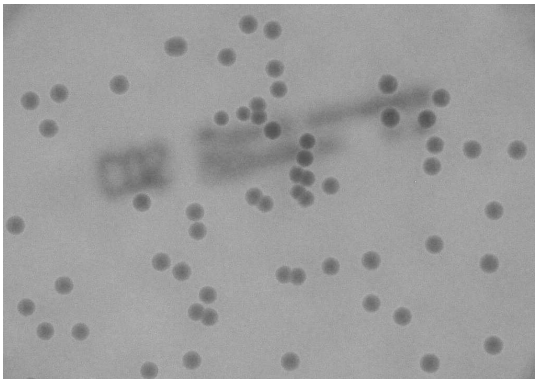
Ein RGB- oder HSV-Farbbild hat die Eigenschaft, dass jedem Pixel ein Farbwert mit drei Dimensionen zugeordnet wird. Berücksichtigt man bei dieser Zuordnung nur einen Farb-Parameter, so kann man das als Grauwert-Bild interpretieren. Dieses enthält dann nur die Informationen des ausgewählten Farbparameters

[1] $\lfloor x \rfloor$ ist die größte ganze Zahl kleiner gleich x

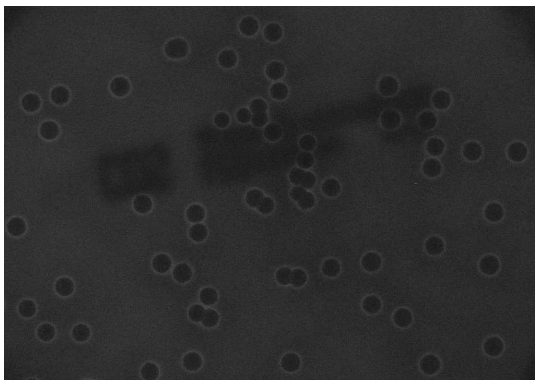
und wird auch als Farbkanal bezeichnet. (vgl. [WIK16b]) Abbildung 8 zeigt ein farbiges Koloniebild und dessen Farbkanäle des RGB- und HSV-Farbraums.



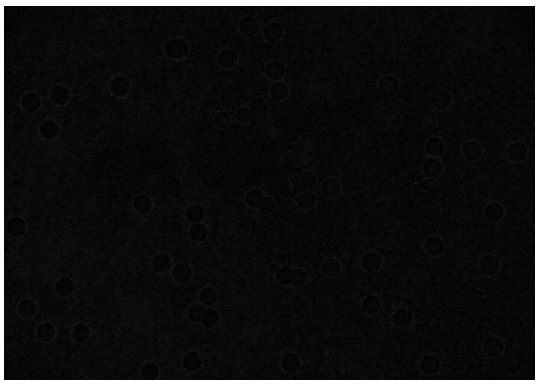
(a) Originalbild



(b) Rot



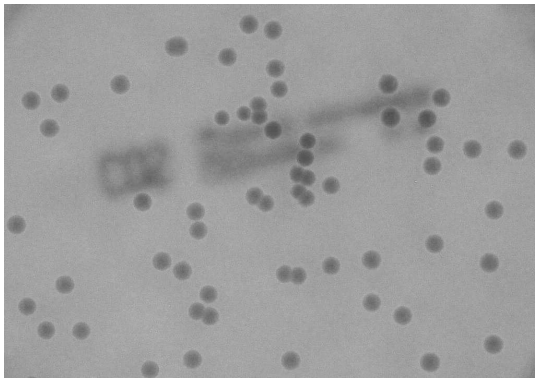
(c) Grün



(d) Blau

(e) Farbton

(f) Sättigung



(g) Helligkeit

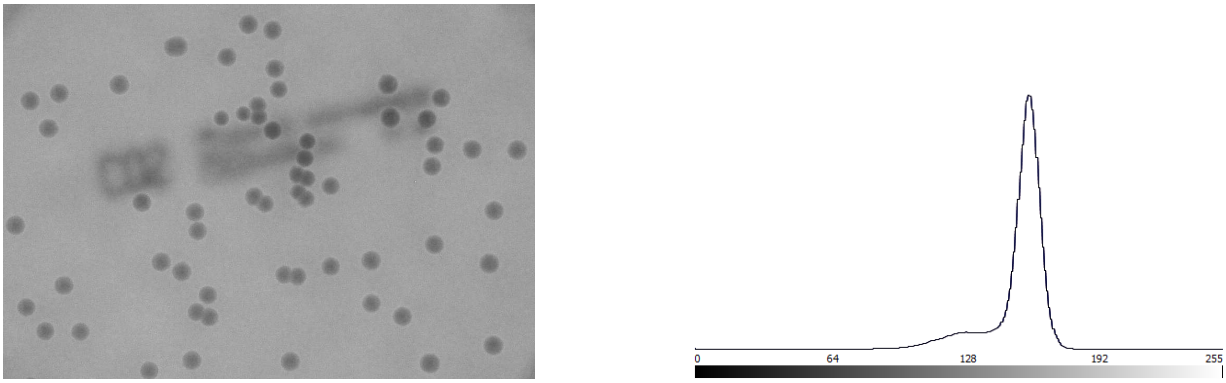
Abbildung 8: Koloniebild und dessen Grauwertbilder aus verschiedenen Farbkanälen

Beim Betrachten dieser Abbildung wird auch deutlich, dass bei einigen Farbkanälen die Unterscheidung zwischen den kreisförmigen Kolonien und dem Hintergrund bei einigen Farbkanälen wesentlich besser gelingt als bei anderen. So tritt der Unterschied zwischen Vordergrund und Hintergrund beim Rot-Kanal des RGB-Farbraums und Helligkeits-Kanals des HSV-Farbraums besonders deutlich zutage, während er beim Blau-Kanal des RGB-Farbraums und dem Farbton-Kanal des HSV-Farbraums kaum noch erkennbar ist. Der Umstand, dass einige Farbkanäle sich bei einem Bild besser für das Unterscheiden von Objekten und Hintergrund eignen als andere, ist gegebenenfalls von Bedeutung für die Bildsegmentierung. An dieser Stelle soll auf den Abschnitt 3 dieser Arbeit verwiesen werden, in dem der eben verdeutlichte Sachverhalt zur Anwendung kommt.

2.3 grundlegende Begriffe und Eigenschaften von Bildern

2.3.1 Histogramm

Ein Histogramm ist im Allgemeinen eine graphische Darstellung der Häufigkeitsverteilung metrisch skalierter Merkmale. (vgl. [WIK16c]) In der Bildverarbeitung werden Histogramme genutzt um bei Graustufenbildern die Verteilung der Häufigkeit des Auftretens von Grauwerten zu veranschaulichen. Abbildung 9 zeigt das Grauwertbild des Farbkanals der Farbe Rot des im Abschnitt 2.2 vorgestellten Beispielbildes und das zugehörige Histogramm in Form eines Liniendiagramms. Auf der horizontalen Achse ist der Wertebereich der möglichen Grauwerte (0 -255) und auf der vertikalen Achse die absolute Häufigkeit des Vorkommens der Grauwerte im Bild abgebildet.



(a) Grauwertbild (b) Histogramm

Abbildung 9: Grauwertbild und zugehöriges Histogramm

Formal kann die absolute Häufigkeit wie folgt definiert werden:

$$h(g) := \sum_{x=1}^{M} \sum_{y=1}^{N} \delta(f(x, y) - g)$$

$h(g)$ ist die absolute Häufigkeit des Vorkommens eines Grauwertes g im gegebenen Grauwertbild. M entspricht der Breite und N der Höhe des Bildes. $f(x, y)$ liefert den Grauwert des Bildes an der Stelle (x, y) . δ ist die diskrete Deltafunktion, auch Kroneckeresch Delta genannt und ist wie folgt definiert.

$$\delta(a) = \begin{cases} 1, & f\ddot{u}r\, a = 0 \\ 0, & f\ddot{u}r\, a \neq 0 \end{cases}$$

2.3.2 Nachbarschaft

Eine Nachbarschaft ist die Bildregion um einen Pixel. Es gibt zwei grundlegende Nachbarschaftskonzepte, die Vierer-Nachbarschaft (auch D-Nachbarschaft) und die Achter-Nachbarschaft. (vgl. [WIK16g]) Abbildung 10 veranschaulicht die eben genannten Nachbarschaftskonzepte.

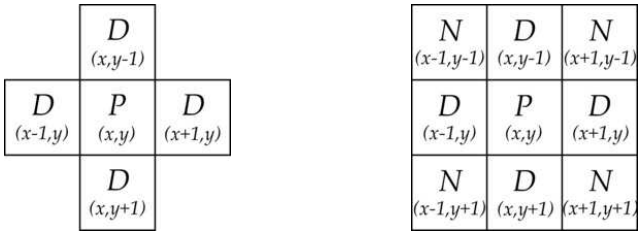
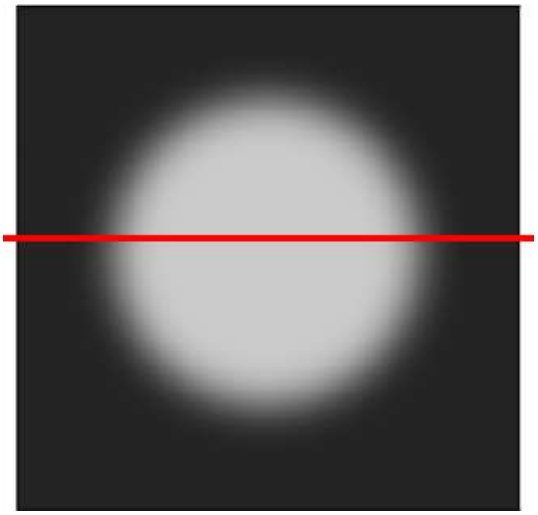


Abbildung 10: a: Vierer-Nachbarschaft um P und b: Achter-Nachbarschaft um P (Quelle: [WIK16g])

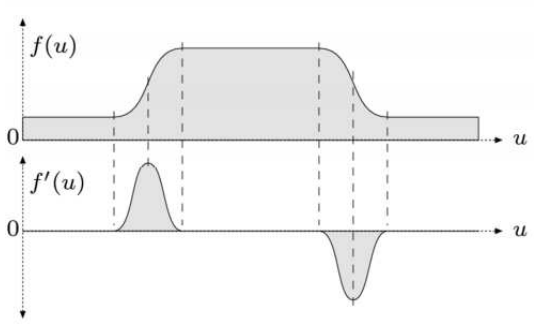
Die Vierer-Nachbarschaft eines Pixels P besteht aus den beiden Pixeln links und rechts von P und den beiden Pixeln ober- und unterhalb von P. Die Achter-Nachbarschaft beinhaltet zusätzlich noch die vier an P angrenzenden Diagonalpixel.

2.3.3 Regionen, Kanten und Konturen

Unter einer Region ist in dieser Arbeit eine zusammenhängende Menge von Pixeln zu verstehen. Unter Kanten oder Kantenpunkten sind die Pixel zu verstehen, die eine Region umschließen und somit von benachbarten Regionen trennen. Für die meisten Menschen sind Kannten intuitiv wahrnehmbar. Eine Eigenschaft von Kantenpunkten, anhand der sie im Bereich der Bildverarbeitung bestimmt werden können, ist, dass sie sich von benachbarten Bildpunkten in ihrem Farbwert stark unterscheiden. Sie sind also dadurch gekennzeichnet, dass in ihrer Umgebung eine große Farbwertänderung auftritt. Diese Änderung im Farbwert kann in horizontaler Richtung und/oder in vertikaler Richtung erfolgen. Abbildung 11 verdeutlicht den horizontalen Verlauf der Farbwerte (Grauwerte) eines horizontalen Schnitts durch ein Grauwertbild. Die horizontal angeordneten Bildpunkte des Schnitts lassen sich als eindimensionale Funktion auffassen, deren Verlauf auf der rechten Seite der Abbildung (b) oben dargestellt ist.



(a) Grauwertbild mit gekennzeichneten horizontalen Verlauf



(b) Funktion und Ableitung

Abbildung 11: Eindimensionaler Schnitt durch ein Grauwertbild (Quelle: [SEB15] S. 3)

Auf dem Bild (a) ist eine helle Kreisscheibe auf dunklem Hintergrund zu sehen. Im Funktionsverlauf zeichnet sich der Hintergrund durch kleine und die Kreisscheibe durch große Funktionswerte (Grauwerte) aus. Von links nach recht betrachtet, zeichnet sich die Funktion am Übergang vom Hintergrund zum Kreis durch einen Anstieg der Funktionswerte aus. Betrachtet man den Funktionsverlauf der Ableitung (b unten), so fällt auf, dass der Anstieg der Funktion zunächst zunimmt, bis er ein Maxima erreicht hat und dann wieder abnimmt. Beim Übergang vom Kreis zum Hintergrund ist der eben beschriebene Funktionsverlauf mit negativem Anstieg zu beobachten. An den Stellen, an denen sich ein lokales Extremum befindet, ist der Betrag des Anstiegs und somit auch die Änderung des Farbwerts lokal am größten. Das ein derartiges Verhalten an einer Stelle in horizontaler, oder vertikaler Richtung auftritt, ist ein notwendiges aber kein hinreichendes Kriterium dafür, dass diese Stelle ein Kantenpunkt ist. Ob es sich bei der betreffenden Stelle tatsächlich um einen Kantenpunkt handelt, hängt davon ab, wie groß der Betrag des Anstiegs an dieser Stelle ist. Die Größe des Schwellwertes des Betrages der Änderung, ab der eine Stelle mit vertikalem oder horizontalem Extremum eine Kante ist, ist vom konkreten Bildmaterial abhängig.

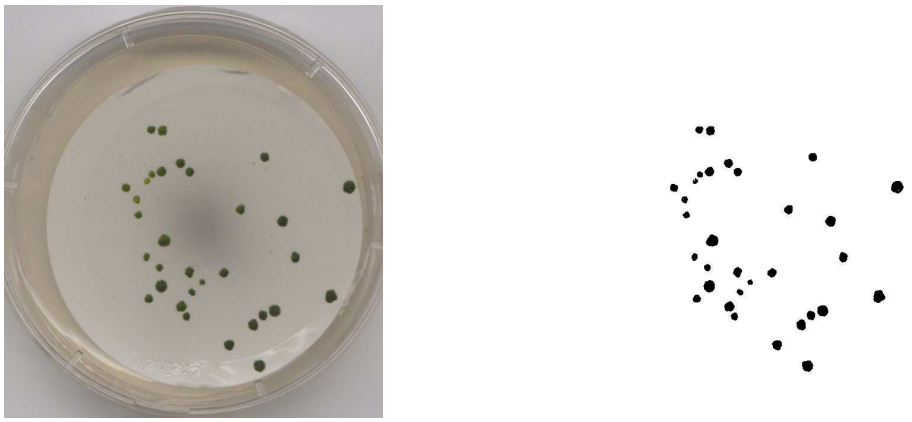
Unter einer Kontur versteht man die Kantenpunkte einer Region, die vollständig an deren Rand verlaufen und eine geschlossene Linie bilden. (vgl. [CD98]S. 111)

3 Bildsegmentierung

3.1 Was ist Bildsegmentierung?

Ein wichtiges Teilgebiet der Bildverarbeitung ist die Bildanalyse. Das Ziel der Bildanalyse besteht darin, durch Anwendung unterschiedlicher Ansätz sowie unterschiedlichen Techniken der Bildtransformation und -manipulationen aus den Low-Level-Pixelinformationen eines Eingabebildes die für weitere Bearbeitungen notwendige symbolische Beschreibungen der Bildinhalte zu ermitteln. (vgl. [TL97] S. 359)

Ein wichtiger Teilschritt der Bildanalyse ist die Bildsegmentierung (Segmentierung). Unter ihr versteht man im weitesten Sinne die sinnvolle Aufteilung eines Bildes in Regionen. In der Literatur findet man verschiedene Definitionen von Bildsegmentierung. In dieser Arbeit wird unter Bildsegmentierung eine Einteilung eines Bildes in Objekt-Regionen (Objekte) und Hintergrundregionen (Bildhintergrund) verstanden. Man spricht bei dieser Art von Segmentierung auch von der einfachen Segmentierung. (vgl. [Erh08] S. 105) Ihr Ergebnis ist ein Binärbild in dem die Pixel, die den Objekten zugeordnet sind, den Farbwert Schwarz und die Pixel, die dem Hintergrund zugeordnet sind, den Farbwert Weiß haben. Abbildung 12 zeigt ein farbiges Eingabebild und das durch einen Segmentierungsprozess erzeugte Ergebnis in Form eines Binärbildes.



(a) Originalbild (b) Ergebnis der Segmentierung

Abbildung 12: Eingabebild und segmentiertes Bild

Für das Segmentieren eines Bildes ist die Eigenschaft, dass die Objekte irgend eine Art von wahrnehmbarer und bestimmbarer Homogenität besitzen, die sie vom Hintergrund unterscheidet, von elementarer Bedeutung. Diese Homogenität kann die Eigenschaften der Farbe, Kontur, Textur oder Form der Regionen betreffen. Man spricht in diesem Zusammenhang auch von einem Homogenitätskriterium, anhand dessen die Zuordnung der Pixel zu den verschiedenen Regionen erfolgt. (vgl. [TL97] S. 359) Die Verfahren der Bildsegmentierung lassen sich anhand der Eigenschaft, die sie benutzen, kategorisieren. So gibt es pixelorientierte, kanten- bzw. konturbasierte, regionbasierte und texturbasierte Verfahren. In den folgenden Abschnitten 3.2 bis 3.5 sollen die Ansätze sowie einige Vertreter der eben aufgeführten Segmentierungsverfahren vorgestellt werden.

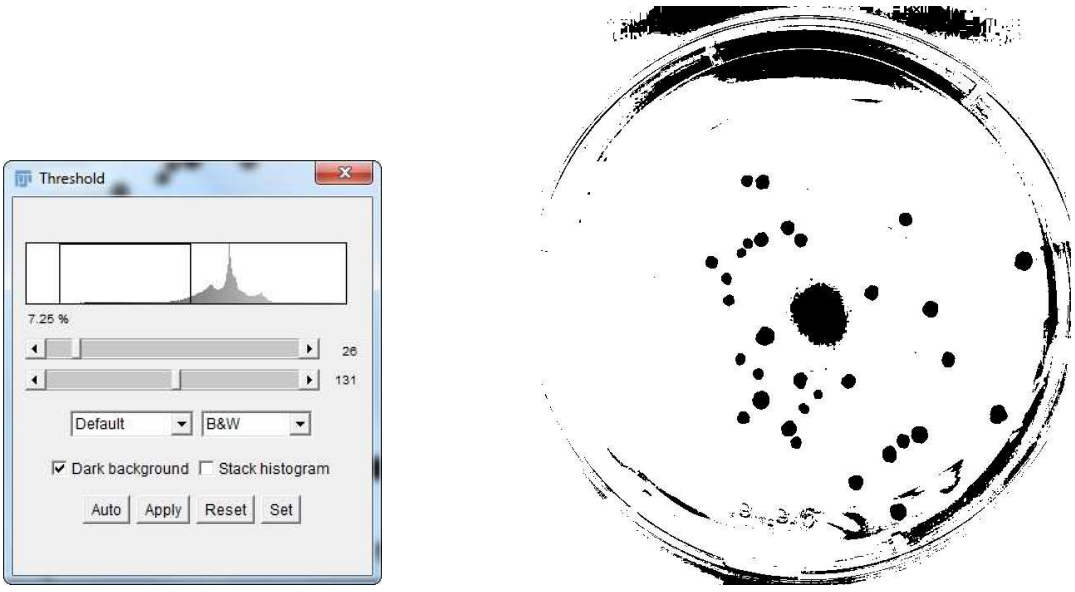
3.2 Pixelorientierte Verfahren

Pixelorientierten Verfahren liegt die Annahme zugrunde, dass sich die Pixel eines Bildes anhand ihrer Farbe Objekten zuordnen lassen und somit bezüglich ihres Farbwertes eine Homogenität aufweisen. Drei Vertreter der pixelorientierten Segmentierungsverfahren, die in dieser Arbeit vorgestellt werden sollen, sind das globale Schwellwertverfahren, das lokale Schwellwertverfahren und das dynamische Schwellwertverfahren. Diese Verfahren bearbeiten Grauwertbilder. Es ist daher notwendig das farbige Eingabebild in ein Grauwertbild zu überführen. Dies geschieht in einem Prozess, der in dieser Arbeit als Vorverarbeitung bezeichnet wird und dessen Verfahren im Abschnitt 3.6 vorgestellt werden.

3.2.1 Globales Schwellwertverfahren

Bei dem globalen Schwellwertverfahren (Global Threshold) wird ein Intervall I [Min, Max] im Grauwert-Farbraum festgelegt, anhand dessen jeder Pixel des Eingabebildes bewertet wird. Min und Max werden auch Schwellwerte genannt. Min ist der kleinste und Max der größte Wert des Intervalls. Ist der Grauwert eines Pixels p in I, dann ist p den Objekten zuzuordnen, andernfalls dem Hintergrund.

Die Bestimmung eines für die Segmentierung des Eingabebild sinnvollen Intervalls ist das Kernproblem des Verfahrens. Sie kann zuverlässig durch einen menschlichen Bearbeiter erfolgen.



(a) Dialog (b) Bild

Abbildung 13: ImageJ Dialog zur Wahl der Schwellwerte und resultierendes Bild

Anhand der Abbildung 13 lässt sich verdeutlichen, wie eine manuelle Wahl der Schwellwerte erfolgen kann. Sie zeigt den Dialog (a) der Bildverarbeitungssoftware *ImageJ* zur manuellen Bestimmung der Schwellwerte und das aus der Anwendung der gewählten Schwellwerte resultierende Ergebnisbild (b). Das globale Thresholdverfahren lässt sich mit moderner Rechentechnik so schnell ausführen, dass dem Benutzer bei der Konfiguration das Ergebnisbild in Echtzeit angezeigt werden kann.

Es existieren aber auch Methoden zur automatischen Bestimmung der Schwellwerte. Ein Beispiel dafür ist das Verfahren von *Otsu*, das nach dem japanischen Mathematiker *Nobuyuki Otsu* benannt wurde. Es folgt der Annahme, dass ein Eingabebild zwei Klassen von Pixeln enthält, die innerhalb ihrer Klasse weitestgehend homogen in ihrer Farbe sind und sich von den Pixeln der anderen Klasse farblich unterscheiden. Aus dieser Eigenschaft lässt sich schlussfolgern, dass die Häufigkeitsverteilung der Grauwerte des Eingabebildes die Eigenschaft der Bimodalität (Zweigipfligkeit) besitzt. (vgl. [WIK16h])

Mit dem Otsu Verfahren wird lediglich der Max-Wert des Intervalls ermittelt. Der Min-Wert wird auf 0-festgelegt. Daraus folgt, dass bei seiner Anwendung bei Bildern mit hellen Objekten auf dunklem Hintergrund ein Binärbild entstehen kann, in dem die Pixel der Objekte den Farbwert für Schwarz und die Pixel des Hintergrundes den Farbwert Weiß haben. Bei dunklen Objekten auf hellem Hintergrund, wird ein Binärbild erzeugt, in dem die Pixel der Objekte und die Pixel des Hintergrundes gegenteilig gefärbt sind. Es muss daher gegebenenfalls noch invertiert werden.

Der Max-Wert wird im Otsu-Verfahren so bestimmt, dass die Streuung innerhalb der dadurch entstehenden Klassen möglichst klein, zwischen den Klassen gleichzeitig aber möglichst groß ist. Dazu wird der Quotient zwischen den beiden Varianzen gebildet und ein Max-Wert gesucht, bei dem dieser möglichst groß wird.

Formal lässt sich das Otsuverfahren wie folgst beschreiben:

Seien $K_0(Max)$ und $K_1(Max)$ zwei Klassen von Punkten die mittels des zu bestimmenden Max-Wertes voneinander getrennt werden. p(g) sei die Auftrittswahrscheinlichkeit des Grauwertes 0 < g < 255. Dann ergibt sich die Wahrscheinlichkeit P_0 und P_1 des Auftretens von Pixeln der beiden Klassen mit:

$$K_0 : P_0(Max) := \sum_{g=0}^{MAX} p(g) \; und \; K_1 : P_1(MAX) := \sum_{g=Max+1}^{255} p(g) \; = 1 - P_0(Max)$$

Sei $\overline{g}$ das arithmetische Mittel der Grauwerte innerhalb des gesamten Bildes und $\bar{g0}$ und $\bar{g1}$ die Mittelwerte innerhalb der Klassen $K_0(Max)$ und $K_1(Max)$, dann ergeben sich die Varianzen innerhalb der beiden Klassen wie folgt:

$$\sigma_0^2(Max) := \sum_{g=0}^{Max} (g - \bar{g_0})^2 * p(g) \; und \; \sigma_1^2(Max) := \sum_{g=Max+1}^{255} (g - \bar{g_1})^2 * p(g)$$

Das Ziel ist es, dass die Varianz der Grauwerte in den Klassen σ_{in}^2 möglichst klein und die Varianz zwischen den Klassen σ_{zw}^2 möglichst groß ist. Der Max-Wert ist also so zu wählen, dass der Quotient $\frac{\sigma_{zw}^2}{\sigma_{max}^2}$ einen maximalen Wert annimmt. Wobei sich σ_{in}^2 und σ_{zw}^2 wie folgt bestimmen lassen:

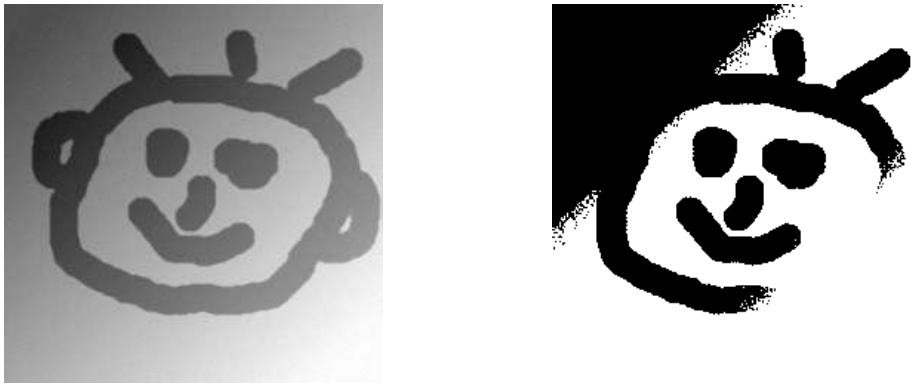
$$\sigma_{zw}^2(Max) := P_0(Max) * (\bar{g_0} - \bar{g})^2 + P_1(Max) * (\bar{g_1} - \bar{g})^2$$

$$\sigma_{in}^2(Max) := P_0(Max) * \sigma_0^2(Max) + P_1(Max) * \sigma_1^2(Max)$$

Neben dem Verfahren von Otsu gibt es noch zahlreiche andere Methoden zur automatischen Bestimmung der Parameter des Schwellwertverfahrens Min und Max. Diese sollen jedoch nicht Gegenstand dieser Arbeit sein.

3.2.2 Lokales Schwellwertverfahren

In vielen Fällen ergibt das globale Schwellwertverfahren ungenügende Ergebnisse. So kann ein Eingabebild beispielsweise einen Helligkeitsverlauf aufweisen, der durch ungleichmäßige Belichtung der fotografierten Szene oder Ähnliches zu Stande kommen kann. Anhand Abbildung 14 lässt sich das daraus entstehende Problem verdeutlichen. Im Ergebnisbild oben links sind die Pixel, die eigentlich zum Hintergrund zugeordnet werden sollten, so dunkel, dass sie zu den Objekten zugeordnet werden. Die Pixel des Objektes unten rechts sind so hell, dass sie fälschlicher Weise dem Hintergrund zugeordnet werden. Bei diesem Bild lassen sich für das globale Schwellwertverfahren keine Min und Max Parameter bestimmen, für die keine fehlerhafte Zuordnung von Pixeln erfolgt.



(a) Eingabebild (b) Ergebnis

Abbildung 14: Anwendung eines globalen Schwellwertverfahren bei einem Bild mit Helligkeitsverlauf (Quelle: [WIK16k])

Abhilfe schafft hier ein Verfahren, das das Bild in mehrere Regionen einteilt und auf ihnen ein Schwellwertverfahren mit lokalen Parametern Min und Max anwendet. Diese Parameter werden mit automatischen Verfahren, wie dem im Abschnitt 3.2.1 vorgestellten Otsu Verfahren bestimmt, indem es auf die jeweilige Region angewendet wird. (vgl. [TL97] S. 365)

3.2.3 Dynamisches Schwellwertverfahren

Eine konsequente Weiterentwicklung des lokalen Ansatzes des lokalen Schwellwertverfahrens ist das dynamische Schwellwertverfahren. Hierbei werden für jedes Pixel des Eingabebildes die Parameter Min und Max berechnet, indem man ein automatisches Verfahren zu deren Bestimmung auf eine quadratische M mal M Pixel große Region um den zu untersuchenden Pixel anwendet. (vgl. [TL97] S. 366)

3.3 Kanten- bzw. konturbasierte Verfahren

Die kantenbasierten Segmentierungsverfahren fußen auf der Grundannahme, dass die Objekte eine homogene, scharfe Berandung besitzen. (vgl. [TL97] S. 367) Wenn Objektregionen sich durch scharfe Berandung auszeichnen, dann können sie prinzipiell durch geschlossene Randkurven beschrieben werden. Hierfür bedarf es geeigneter Verfahren zur Ermittlung von Kantenpunkten und Verfahren, die aus den gefundenen Kantenpunkten Randkurven erzeugen, die diese Objekte umschließen. Aus den Randkurven können dann beispielsweise Polygone für die Beschreibung der Objekte erzeugt werden.

3.3.1 Bestimmen von Kantenpunkten

Wie im Abschnitt 2.3.3 erörtert, zeichnet sich ein Kantenpunkt dadurch aus, dass in seiner Umgebung eine signifikante Änderung der Grau- oder Farbwerte erfolgt. Bei der Auffassung eines Bildes als zweidimensionale Funktion (siehe Abschnitt 2.1) bedeutet das, dass der Gradient an der Stelle des Kantenpunktes signifikant groß ist. Darüber hinaus besitzt die Ableitung des Bildes eine Nullstelle. Verfahren zur Bestimmung von Kantenpunkten machen sich die eben beschriebenen Eigenschaften der ersten oder der zweiten Ableitung zu Nutze. An dieser Stelle soll ein Verfahren namens *Sobel-Operator*, das auf der ersten Ableitung basiert, vorgestellt werden.

Bei dem Sobel-Operator wird der Gradient (Gx , Gy) des zu untersuchenden Punktes numerisch bestimmt. Gx ist der Wert der partiellen Ableitung in horizontaler und Gy ist der Wert der partiellen Ableitung in vertikaler Richtung. Für die Bestimmung von Gx und Gy wird das Eingabebild A an der Stelle des Punktes mit den Sobel-Operatoren *Sx* und *Sy* gefaltet:

$$Gx := Sx * A := \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} * A$$

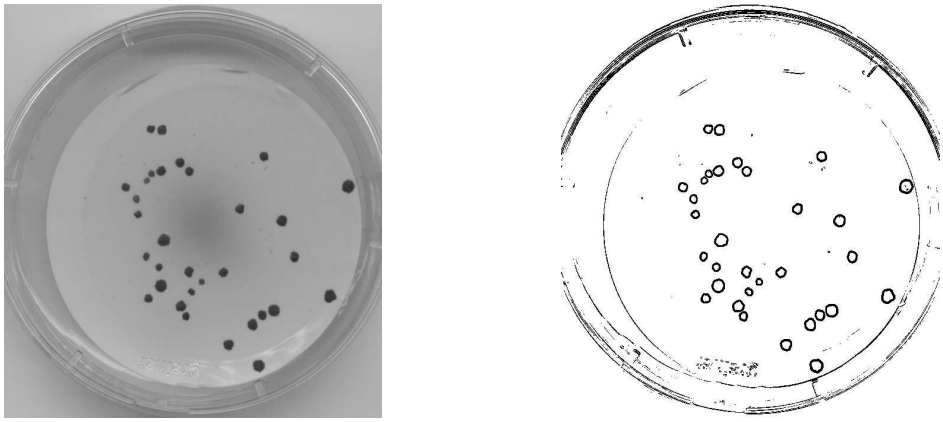
$$Gy := Sy * A := \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * A$$

Das Prinzip der Faltung wird im Abschnitt 3.6.1 dieser Arbeit genauer beschrieben.

Bildet man nun den Betrag G des Gradienten, so hat man ein Kriterium für die Bewertung eines Punktes bezüglich seiner Kanteneigenschaft gefunden.

$$G = \sqrt{Gx^2 + Gy^2}$$

Die Klassifizierung eines Punktes als Kantenpunkt kann mit Hilfe eines Schwellwertes, den der Gradientenbetrag G an der Stelle des Punktes überschreiten muss, um den Kantenpunkten zugeordnet zu werden, erfolgen.



(a) Eingabebild (b) Kantenbild

Abbildung 15: Eingabebild und Kantenbild

Mit Hilfe des Sobel-Operators kann aus einem Eingabebild ein Kantenbild erzeugt werden. Dafür muss der Sobel-Operator auf jeden Pixel des Bildes angewendet werden. Abbildung 15 zeigt ein Eingabebild und das unter Anwendung des Sobel-Operators extrahierte Kantenbild.

3.3.2 Konturenverfolgung

Eine komplexe Problemstellung der kantenbasierten Segmentierung ist das Zuordnen von Kantenpunkten zu Konturen. Diese Problematik lässt sich in zwei Schritte unterteilen:

- das Finden geeigneter Startpunkte für die Linienverfolgung
- das Auffinden von Nachfolgepunkten

Die einfachste Möglichkeit zur Auffindung von Startpunkten ist die interaktive Methode, bei der der Benutzer den Startpunkt manuell wählt. Eine automatisierte Wahl von Startpunkten, kann beispielsweise dadurch erfolgen, dass Bildpunkte anhand bestimmter Merkmale als Startpunkt klassifiziert werden. Beispielsweise können bei Graustufenbildern Bildpunkte in einen zweidimensionalen Merkmalsraum, bestehend aus Grauwert und Betrag des Gradienten, abgebildet werden. (vgl. [TL97] S. 372)

Die Bestimmung von Nachfolgepunkten kann wie folgt geschehen:

Angenommen die Punkte z_{i-2}, z_{i-1}, z_i wurden bereits als auf der Kontur liegend erkannt, dann gilt es nun den nächsten benachbarten Konturpunkt z_{i+1} zu ermitteln. Das kann mit der Hilfe von sogenannten Suchstrahlen erfolgen. Ein Suchstrahl ist ein gerader Pixelpfad mit fester Länge, der in einem bestimmten Winkel zur Linie zwischen z_{i-1} und z_i an den Punkt z_i gelegt wird. Entlang der Suchstrahlen werden nun mögliche Nachfolgepunkte gesucht. Das dafür angewendete Bewertungskriterium kann zum Beispiel eine Ähnlichkeit des Farbwerts zum Farbwert von z_i sein. Abbildung 16 veranschaulicht die Nachfolgerbestimmung.

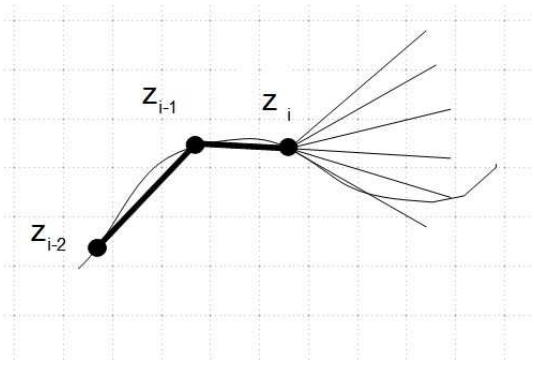


Abbildung 16: Darstellung der Nachfolgerbestimmung

3.3.3 Hough-Transformation

Ausgangspunkt für die *Hough-Transformation* ist ein aus dem Eingabe-Bild ermitteltes Kantenbild. Voraussetzung für die Hough-Transformation ist, dass sich die bestimmenden Objekte als parametrisierbare geometrische Gebilde beschreiben lassen. So eignet sich die Hough-Transformation beispielsweise für Bilder mit kreisähnlichen Objekten.

"Das Grundprinzip der Transformation ist schnell erklärt, es ist ähnlich wie bei einer Wahl:

- Erstelle ein Rasterfeld möglicher Parameterwerte (Die Politiker, die sich zur Wahl stellen)
- Jeder Kantenpunkt gibt eine Stimme für einen Kandidaten im Raster und erhöht diesen (Stimmenanzahl)
- Bestimmte lokale oder globale Maxima im Raster (Wer hat gewonnen?)" [SCH16]

Es wird also ein sogenannter *Hough-Raum* untersucht, der alle möglichen Wertkombinationen von geometrischen Parametern (wie Mittelpunkt, Radius) enthält. Dabei entspricht jeder Punkt im Hough-Raum, einer geometrischen Figur im Eingangsbild und wird mit der Anzahl der Kantenpunkte, die sich auf ihr befinden, bewertet. Ist die Anzahl der Kantenpunkte groß genug, so beschreibt die Parameterkombination ein gesuchtes Objekt.

3.4 Regionenbasierte Verfahren

Der grundlegende Ansatz der regionenbasierten Segmentierungsverfahren ist, dass benachbarte Pixel anhand eines Distanzmaßes zu Regionen zusammengefasst werden.

Ein typisches Distanzmaß ist der Grauwertabstand d von zwei Pixeln p1 und p2. Er wird durch die Differenz der Grauwerte g1 und g2 der beiden Pixel gebildet.

$$d(p1,\, p2) := |g1 - g2|$$

Dabei seien g1 und g2 die Grauwerte der Pixel p1 und p2.

Ein Distanzwert wird für die Beurteilung benötigt, um zu ermitteln, ob zwei benachbarte Regionen ähnlich genug sind, um sie zu einer Region zusammenzufassen. Für diese Beurteilung gibt es mehrere Strategien, die auch *Linkage-Strategien* genannt werden.

Eine Beispiel für diese Linkage-Strategien ist die sogenannte *Single-Linkagestrategie*. (vgl. [TL97]S. 376) Sie lässt eine Region A und eine Region B miteinander verschmelzen, wenn es ein benachbartes Pixelpaar p_A, p_B mit $p_A \in A$ und $p_B \in B$ gibt, so dass gilt:

$$d(p_A,\, p_B) \leq t$$

Dabei ist t ein zu definierender Schwellwert für den Grauwertabstand.

Im Folgenden sollen zwei regionenbasierte Segmentierungsverfahren vorgestellt werden, bei denen Distanzmaße und Linkage-Strategien zur Anwendung kommen.

3.4.1 Region-Growing

Das Bereichswachstumsverfahren (engl. region growing) besteht aus zwei Schritten:

Im ersten Schritt werden sogenannte Keimpunkte gewählt, die als Anfangsregionen betrachtet werden. Alle anderen Pixel werden als nicht zu einer Region zugehörig klassifiziert.

Im zweiten Schritt werden die Pixel in Umgebung der Regionen unter Verwendung eines Distanzmaßes bewertet. Wenn das Ergebnis der Distanzfunktion unter einem gewissen Schwellwert liegt, dann wird der Pixel als der zu untersuchenden Region zugehörig klassifiziert. Befindet sich der betrachtete Pixel bereits in einer anderen Region, werden beide Regionen mittels einer Linkage-Strategie bewertet und gegebenenfalls miteinander verschmolzen.

Die Wahl der Keimpunkte ist für die Funktionsweise des Verfahrens von großer Bedeutung. (vgl. [TL97]S. 378) Diese kann entweder manuell durch einen Benutzer erfolgen, oder automatisiert ermittelt werden. Ziel des Verfahrens ist die Herausbildung von möglichst homogenen Regionen. Daher ist es sinnvoll, die Keimpunkte in möglichst homogene Bereiche des Eingabebildes zu legen. Dafür kann beispielsweise ein Minimum des Gradienten als Indikator für hohe Homogenität dienen.

3.4.2 Region-Merging

Im Gegensatz zum Region-Growing-Verfahren wird im Region-Merging-Verfahren im Startzustand jeder Pixel als Region betrachtet. Im Iterationsschritt werden benachbarte Regionen paarweise mit einem Distanzmaß betrachtet und gegebenenfalls zu einer neuen Region zusammengefasst. Das Verfahren terminiert, wenn keine Verschmelzung von Regionen mehr erfolgt. (vgl. [TL97]S. 378)

3.5 Texturbasierte Verfahren

Die grundlegende Annahme für texturbasierte Segmentierungsverfahren ist, dass sich Regionen durch eine Homogenität in ihrer Textur auszeichnen. Abbildung 17 zeigt verschiedene Texturen, die sich markant voneinander unterscheiden. Für den Textur-Begriff bietet die Literatur eine Vielzahl von Beschreibungsversuchen, die jedoch stark auf konkrete Anwendung zugeschnitten sind. (vgl. [TL97]S. 383)

Ein möglicher Zugang zum Textur-Begriff besteht in der Identifikation verschiedener Eigenschaften, die bei der Wahrnehmung und Unterscheidung von Texturen eine Rolle spielen. Beispiele für diese Eigenschaften sind Periodizität, Komplexheit, oder Gerichtetheit. (vgl. [TL97]S. 384)

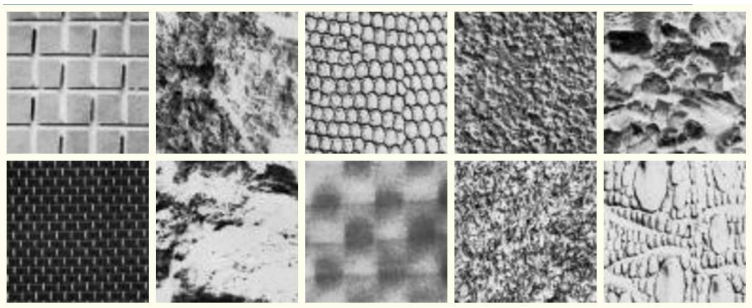


Abbildung 17: Typische Texturen aus dem Album von Brodatz, wie sie zum Test von Verfahren benutzt werden (Quelle: [BRO])

Eine Möglichkeit der Quantifizierung dieser Eigenschaften von Texturen bietet die statistische Texturanalyse. In ihr wird die Häufigkeitsverteilung der Grauwerte von Bildausschnitten (Fenster) ausgewertet. Daraus ergeben sich statistische Maße wie Mittelwert, Varianz oder Schiefe, die als Maße für die Texturerkennung (Texturmaße) dienen können.

Die Vorgehensweise für texturbasierte Segmentierungsalgorithmen lassen sich in 2 Schritte gliedern:

1. Für jedes Pixel werden sogenannte Features berechnet. Das sind n-dimensionale ¨ Vektoren, deren Komponenten beispielsweise einzelne Texturmaße sind. Die Feature-Vektoren spannen einen Feature-Raum auf.

2. Die Untersuchung des Feature-Raumes erfolgt mittels geeigneter Metriken. Dabei wird der Feature-Raum in Gruppen zerlegt, die die einzelnen Regionen repräsentieren.

3.6 Vorverarbeitung

Die Vorverarbeitung umfasst Verfahren zur Bildverbesserung. Ihr Ziel ist es das Eingabebild für die nachfolgenden Bearbeitungsschritte aufzubereiten. Welche Techniken dabei angewendet werden, hängt von den Eigenschaften des Eingabebildes und den angewendeten Verfahren der nachfolgenden Schritte ab.

Viele der bisher vorgestellten Segmentierungsverfahren arbeiten mit Grauwertbildern. Ist das Eingabebild ein Farbbild, so muss es für die Anwendung eines dieser Segmentierungsverfahren zu einen Grauwertbild transformiert werden. Dies kann beispielsweise durch die Wahl eines geneigten Farbkanals passieren. Wie im Abschnitt 2.2.6 dieser Arbeit anhand eines Beispieles (Abbildung 8) bereits verdeutlicht wurde, lassen sich Objekte in einigen Farbkanälen eines Farbbildes wesentlich besser erkennen als in anderen.

Die Vorverarbeitung umfasst aber auch Techniken zur Verminderung von Bildrauschen, Eliminierung kleinerer Strukturen (Homogenisierung), und zur Korrektur von ungleichmäßiger Beleuchtung. Einige dieser Techniken sollen im Folgenden vorgestellt werden.

3.6.1 Filter

Filter-Operatoren erzeugen aus einem Eingabebild ein Ergebnisbild, indem für die Berechnung der Pixel im Ergebnisbild die Pixel einer definierten Umgebung des Eingabebildes mit einbezogen werden. Oft entspricht die Umgebung der im Abschnitt 2.3.2 vorgestellten Achter-Nachbarschaft. Sie kann aber auch beliebig groß sein. Abbildung 18 verdeutlicht die Wirkweise von Filteroperationen anhand einer schematischen Darstellung.

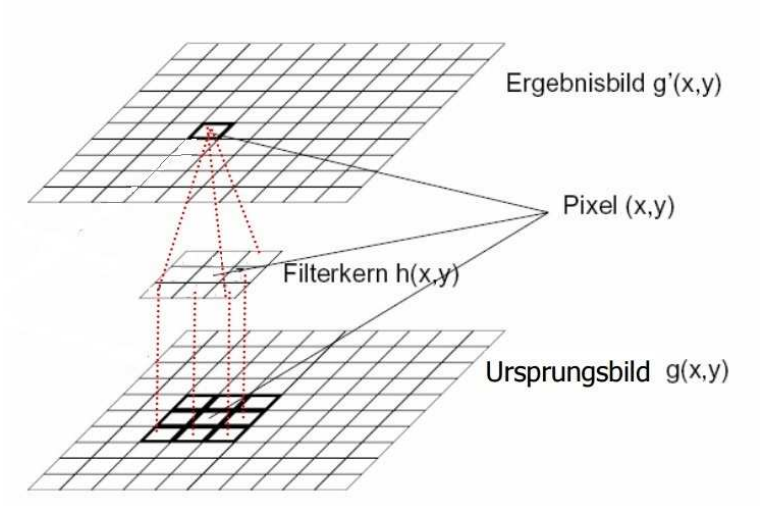


Abbildung 18: Darstellung der Anwendung von Filter (Quelle: [FIL])

Es gibt zwei Arten von Filtern, nämlich lineare Filter und nicht lineare Filter. Bei linearen Filtern berechnet sich jeder Pixel als gewichtete Summe der Pixel einer Umgebung. Derartige Operationen werden auch Faltung genannt. Die Gewichte und auch die zu betrachtende Umgebung werden durch einen Filterkern – auch Filtermaske genannt – definiert. Dabei entspricht ein Gewicht h(i, j) einem Koeffizienten

der Filtermaske:

$$\begin{bmatrix} h(-1,-1) & h(-1,\,0) & h(-1,\,1) \\ h(\,0,-1) & h(\,0,\,0) & h(\,1,\,0) \\ h(\,1,-1) & h(\,1,\,0) & h(\,1,\,1) \end{bmatrix}$$

Die Filtermaske wird so auf das Eingabebild gelegt, dass der zu betrachtende Pixel an der Stelle (i, j)=(0, 0) der Filtermaske ist. Das Ergebnisbild s berechnet sich durch Anwendung der Filtermaske M auf das Eingabebild f wie folgt:

$$s(x,\,y) = \sum_{(i,j)\in M} f(x+i,\,y+i) * h(i,j)$$

Im Abschnitt 3.3.1 kamen bereits lineare Filter für die Kantenextraktion zur Anwendung. Ein anderes Anwendungsgebiet linearer Filter ist das Glätten von Bildern. Zwei dafür geeignete Filter sind der Mittelwert-Filter und der Gauß-Filter.

Bei dem Mittelwert-Filter wird jedes Pixel durch den Mittelwert seiner Umgebungspixel ersetzt. Dies geschieht in dem jede Stelle der Filtermaske mit $\frac{1}{Anzahl\,Pixel\,der\,Filtermaske}$ gewichtet wird. Die Anwendung eines Mittelwert-Filters hat zur Folge, dass nicht nur Rauschen glättet wird, sondern auch Kanten im Bild verwischen.

Bei dem Gauß-Filter wird für die Erzeugung der Filtermaske eine diskrete Version einer diskretisierten zweidimensionalen Normalverteilung benutzt. Eine mit ihr erzeugte Filtermaske hat beispielsweise die folgende Gestalt:

$$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

Der Gauß-Filter hat gegenüber dem Mittelwert-Filter den Vorteil, dass Kanten weniger verwischt werden. (vgl. [TL97] S. 210)

Nicht-lineare Filter sind dadurch ausgezeichnet, dass sie nicht durch Faltung beschrieben werden können. Einer ihrer Vertreter, der zur Rauschverminderung benutzt wird, ist der Median-Filter.

Bei dem Median-Filter wird der Grauwert eines Pixels durch den Median aller Grauwerte der Umgebungspixel ersetzt. Dazu werden die Grauwerte in der Umgebung des Pixels sortiert und aus der so entstandenen Folge der mittlere Wert (den Median) gewählt.

3.6.2 Shading-Korrektur

Wie im Abschnitt 3.2.1 beschrieben wurde, wirken sich Helligkeitsverläufe in Bildern, wie sie durch Aufnahmen mit ungleichmäßiger Ausleuchtung entstehen können, störend auf die Anwendung globaler Schwellwertverfahren aus. Abhilfe können hier Techniken zum Helligkeitsausgleich (Shading-Korrektur) schaffen.

Eine einfache Vorgehensweise der Shading-Korrektur ist die Subtraktion eines Referenzbildes, das den Helligkeitsverlauf des Eingabebildes enthält, vom Eingabebild mit anschließender Grauwertanpassung. (vgl. [TL97] S. 366)

Bei der Subtraktion zweier Grauwertbilder a und b berechnet sich das Ergebnisbild s wie folgt:

$$s(x, y) = a(x, y) - b(x, y)$$

Ein geeignetes Referenzbild kann man beispielsweise durch die Anwendung eines sehr großen Mittelwertfilters z.B. 31x31 auf dem Eingabebild erzeugen.

Abbildung 19 verdeutlicht die eben beschriebene Technik der Shading-Korrektur, anhand eines Beispieles.

(a) Eingabebild

(b) Anwendung eines 31x31 Mittelwertfilter

(c) Subtraktion

(d) Helligkeitsanpassung

Abbildung 19: Anwendung der Shading-Korrektur

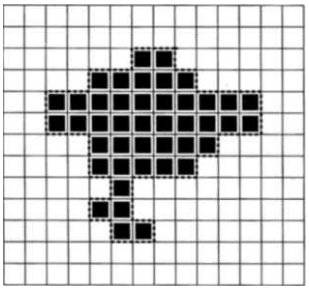
3.7 Nachbearbeitung und Kombination von Segmentierungsverfahren

3.7.1 Erosion und Dilatation

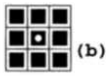
Erosion und Dilatation sind Verfahren zur Beseitigung von Rauschen von binären Ergebnisbildern aus Segmentierungsverfahren. Unter Rauschen ist in diesem Fall das Vorhandensein kleiner isolierter weißer Flächen in den Objektregionen, oder kleine isolierte schwarze Flächen in der Hintergrundregion zu verstehen.

Erosion und Dilatation sind sogenannte morphologische Operatoren, die die Form der Objekte verändern.

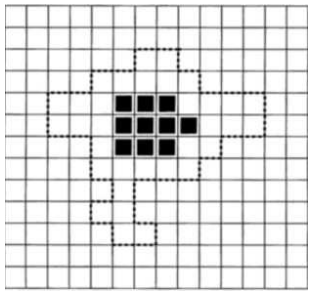
Bei der Erosion wird jeder Pixel des Binärbildes mit einem Strukturelement bewertet. Dabei wird für die Umgebung des jeweiligen Pixels überprüft, ob sie im Strukturelement enthalten ist. (vgl. [TL97] S. 220) Ist dies der Fall, so wird dem Pixel im Ergebnisbild der Farbwert Schwarz, andernfalls der Farbwert Weiß zugeordnet. Erosion bewirkt ein Schrumpfen der Objekte. Abbildung 20 verdeutlicht den Prozess der Erosion anhand eines Beispiels.



(a) Eingabebild



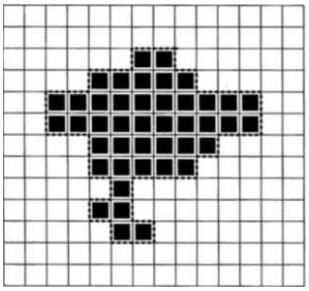
(b) Strukturelement



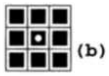
(c) Ergebnis

Abbildung 20: Erosion

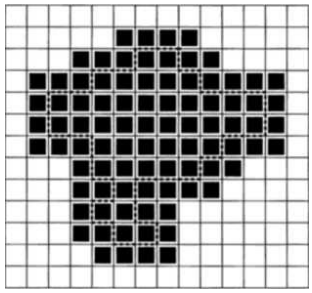
Bei der Dilatation wird ebenfalls jeder Pixel des Binärbildes mit einem Strukturelement bewertet. Doch im Gegensatz zur Erosion wird die Umgebung jedes Pixels dahingehend überprüft, ob ein Pixel der Umgebung im Strukturelement enthalten ist. Dilatation bewirkt ein Wachsen der Objekte. Abbildung 21 verdeutlicht den Prozess der Dilatation anhand eines Beispiels.



(a) Eingabebild



(b) Strukturelement



(c) Ergebnisbild

Abbildung 21: Dilatation

Die Eliminierung des Rauschens, kann unter Hintereinanderausführung von Erosion und Dilatation erfolgen. Dazu wird zunächst der Erosions-Operator so oft auf das Bild angewendet, bis die Rauschregionen komplett verschwunden sind. Danach wird der Dilatations-Operator so oft auf das Bild angewendet, wie vorher der Erosions-Operator, um die Objekte nahezu wieder in ihre Ausgangsform wachsen zu lassen.

3.7.2 Boolesche Operatoren auf Binärbildern

Zwei Binärbilder lassen sich mit booleschen Operatoren wie Und und Oder verknüpfen. Gegeben seien zwei Binärbilder A und B, die mit einen booleschen Operator zu einem Ergebnisbild E verknüpft werden sollen.

Der Und-Operator färbt jedes Pixel im Ergebnisbild schwarz, der auch in A und B schwarz ist. Dies ist beispielsweise bei Binärbildern sinnvoll, die Ergebnisse aus zwei Segmentierungsverfahren sind und die die folgenden Eigenschaften haben: Bei beiden Bildern sind die zu bestimmenden Objektregionen Schwarz gefärbt. Bei A sind aber auch einige Regionen schwarz gefärbt, die dem Hintergrund zuzuordnen sind. Gleiches trifft auf B zu, nur hier sind es andere fehl-klassifizierte Regionen als bei A.

Durch die Anwendung des Und-Operators werden die fehl klassifizierten Regionen aus dem Ergebnisbild eliminiert.

Der Oder-Operator färbt jedes Pixel im Ergebnisbild schwarz, der auch in A oder in B schwarz ist. Dies ist sinnvoll bei zwei Segmentierungsverfahren, die beide jeweils nur eine Teilmenge der Objekte korrekt klassifizieren und sich mit der Oder-Verknüpfung gegenseitig ergänzen können.

4 Segmentierungsfehlerminimierung als neuer Ansatz für die Bildsegmentierung

4.1 Grundidee: Klassen, Samples und Fehlerbretrachtung

Wie im vorangegangenen Abschnitt 3 deutlich wurde, gibt es eine Vielzahl verschiedener Segmentierungsverfahren, -algorithmen, -operationen und -ansätze, die sich für verschiedene Anwendungsfälle eignen. Wie im Abschnitt 1 dargestellt und begründet wurde, ist das Grundanliegen dieser Arbeit ein Verfahren zu entwickeln, das die Segmentierung aus der Sicht des Benutzers vereinfacht. In diesem Abschnitt soll dieses Verfahren herausgearbeitet und vorgestellt werden.

Die grundliegende Idee des Verfahrens ist es, dem Benutzer einen Mechanismus zur Verfügung zu stellen, mit dem dieser das für die Segmentierung benötigte A-Priori Wissen intuitiv definieren kann, und anhand dessen dann verschiedene Segmentierungsalgorithmen konfiguriert und auf ihre Eignung für das zu segmentierende Bildmaterial geprüft werden. Der Benutzer definiert das A-Priori Wissen, indem er zunächst für das zu segmentierende Bild *Segmentklassen (Klassen)* anlegt und diesen anschließend beispielhafte *Regionen (Samples)* des Bildes zuordnet.

Definition 1. Sample:
Ein Sample ist eine durch eine geometrische Figur (z.B. Kreisfläche, Rechteckfläche, Polygonfläche) definierte Teilmenge der Pixel eines Bildes.

Definition 2. Klasse:
Eine Klasse repräsentiert eine Menge von Regionen im Bild, die in sich und untereinander eine Homogenität aufweisen. Sie ist definiert durch einen Bezeichner und einer Menge von ihr zugeordnetten Samples.

Für ein zu segmentierendes Bild sind mindestens zwei Klassen anzulegen, nämlich eine Klasse für die Hintergundregion/en mit dem zwingenden Bezeichner *"background"* und eine Klasse für die Objektregionen mit einem frei wählbaren Bezeichner. Darüberhinaus können Klassen für weitere Objektregionen definiert werden. Das macht beispielsweise dann Sinn, wenn es in einem Bild mehrere Arten von Objekten gibt und es für jede Art von Objekt segmentiert werden soll.

Anhand der definierten Klassen mit Samples werden im nächsten Schritt verschiedene Bildverarbeitungsalgorithmen möglichst optimal konfiguriert und hinsichtlich ihrer Eignung für die Segmentierung bewertet. Das dazu herangezogene Bewertungskriterium ist der *Segmentierungsfehler*. Im Folgenden soll das entwickelte Verfahren deshalb *SM-Verfahren (Segmentierungsfehlerminimierungsverfahren)* genannt werden.

Definition 3. Segmentierungsfehler $E_{segment}$:
Seien $C_1, ...C_i..., C_n$ für ein Eingabebild definierte Segmentierungsklassen. $P_i := \{p_{i1}, p_{i2}, ..., p_{im}\}$ sei die Menge der durch die Sampel der Klasse C_i definierten Bildpunkte. $\hat{P}_i := \{\hat{p_{i1}}, \hat{p_{i2}}, ..., \hat{p_{ik}}\}$ sei die Menge der durch die Anwendung eines Segmentierungsalgorithmus Alg der Klasse C_i zugeordneten Bildpunkte. $\bar{P}_i$ sei die Menge der Bildpunkte, die mit den Samples der Klasse C_i definiert wurden, aber ihr mit dem Segmentierungsalgorithmus nicht zugeordnet wurden sind.

$$\bar{P}_i := \{p \mid p \in P_i \wedge p \notin \hat{P}_i\}$$

$\tilde{P}_i$ sei die Menge der Bildpunkte die in in den Samples einer anderen Klasse als C_i definiert wurden, aber mit dem Segmentierungsalgorithmus C_i zugeordnet wurden sind.

$$\tilde{P}_i := \{p \mid p \in P_j \wedge p \in \hat{P}_i\},\ i \neq j$$

Dann ist der Segmentierungsfehler $E_{segment}$, der sich durch die Ausführung von Alg für C_i ergibt, definiert durch die Summe des Fehlers 1. Art E_{1ART} und des Fehlers 2. Art E_{2ART}:

$$E_{segment} := E_{1ART} + E_{2ART}$$

mit

$$E_{1ART} := \frac{|\bar{P}_i|}{\sum_{gesammt}}$$

und

$$E_{2ART} := \frac{\left|\tilde{P}_i\right|}{\sum_{gesammt}}$$

wobei $\sum_{gesammt}$ die Anzahl der in allen Klassen durch Samples definierten Pixel ist.

$$\sum gesammt := \sum_{i=0}^{n} |P_i|$$

Das SM-Verfahren konfiguriert Bildsegmentierungsalgorithmen für die jeweilige Zielklasse so, dass der Segmentierungsfehler nach ihrer Ausführung möglichst minimal ist. Der minimale Segmentierungsfehler des jeweiligen Algorithmus kann dann als Maß dafür dienen, welcher Segmentierungsalgorithmus am besten für die Segmentierung des jeweiligen Bildmaterials geeignet ist. Um zu verdeutlichen, wie dies genau umgesetzt wird, soll zunächst im Abschnitt 4.2 ein formales Modell für Segmentierungsalgorithmen vorgestellt werden. Im Anschluss daran wird im Abschnitt 4.3 ein Modell vorgestellt, mit dem man Instanzen des Algorithmen-Modells um Informationen zur Analyse des mit ihm beschriebenen Segmentierungsalgorithmus erweitern kann. Im Abschnitt 4.4 werden schließlich auf den beiden Modellen basierende Algorithmen zur Ausführung und Analyse von Segmentierungsalgorithmen beschrieben.

4.2 Algorithmengraph als Modell für Segmentierungsalgorithmen

Möchte man ein Bild auf herkömmlichem Wege segmentieren, dann könnte ein möglicher Ablauf dafür wie folgt aussehen:

Der erste Schritt könnte beispielsweise die Extraktion verschiedener Farbkanäle aus dem Bild sein, von denen man dann einen geeigneten für die weitere Bearbeitung auswählt. Im zweiten Schritt könnte dann ein Median-Filter mit einer geeigneten Filtermatrix auf dem gewählten Bild ausgeführt werden, um gegebenenfalls vorhandenes Rauschen zu eliminieren. Auch das Ausführen einer Shading-Korrektur-Operation zur Beseitigung von Helligkeitsverläufen wäre denkbar. Anschließend könnte dann eine globale Threshold-Operation auf das Ergebnisbild der Shading-Korrektur-Operation oder der Median-Filter-Operation angewendet werden um ein binäres Bid mit den vom Hintergrund getrennten Objekten zu erhalten. Abschließend könnte nun eine Nachbearbeitung erfolgen, um das Ergebnis der Threshold-Operation zu verbessern. Dafür könnten beispielsweise die Erosion- und Dilatation-Operationen auf das Binärbild angewendet werden, um gegebenenfalls vorhandene und unerwünschte Inhomogenitäten zu beseitigen.

Aus dem eben geschilderten möglichen Ablauf von Bildverarbeitungs-Operationen zur Segmentierung eines Bildes, lässt sich eine Vorgänger/Nachfolger-Beziehung zwischen den Operationen ableiten:

Definition 4. Vorgänger/Nachfolger-Beziehung:
Ist das Ergebnis der Ausführung eines Operators o1 Eingabe eines Operators o2, dann heißt o1 Vorgänger von o2 und o2 heißt Nachfolger von o1.

Betrachtet den eben dargestellten Ablauf, dann könnte man daraus schlussfolgern, dass ein Segmentierungsvorgang eine Sequenz von hintereinander ausgeführten Operationen der Bildverarbeitung ist. Doch wie aus der Existenz der im Abschnitt 3.7.2 vorgestellten boolischen-Operationen hervorgeht, können auch parallele Abläufe von Operationen mit anschließender Verknüpfung sinnvoll sein. Für die Vorgänger/Nachfolgerbeziehung bedeutet das, dass eine Operation der Nachfolger von mehreren Operationen sein kann. Denkbar ist auch, dass eine Operation Vorgänger von mehreren Operationen ist. Ein Beispiel hierfür wäre eine *Splitchannel*-Operation, die aus einem Farbbild mehrere Farbkanäle extrahiert, die dann wiederum als Eingabe für mehrere Filter-Operationen fungieren können.

Eine für die Modellierung von Operationen und deren Vorgänger/Nachfolger-Beziehungen geeignete abstrakte Struktur ist ein gerichteter Graph. So könnten Operationen als Menge von Knoten und ihre Vorgänger/Nachfolger-Beziehung als Menge geordneter Knotenpaare also Kanten aufgefasst werden. Doch sind auch weitere Eigenschaften der Verknüfungen von Operationen und des Ablaufes einer Bildsegmentierung zu berücksichtigen, die mit einem einfachen gerichteten Graphen nicht modelliert werden können. Diese Eigenschaften sind:

1. Es gibt eine Start-Operation, deren Eingabe das zu segmentierende Bild ist.
2. Es gibt eine End-Operation, deren Ausgabe das segmentierte Bild ist.
3. Es muss bei einer Operation o1 mit mehreren Ausgaben festgelegt werden, welche Nachfolgeoperation welcher Ausgabe von o1 zugordnet wird und als welche Eingabe der Nachfolgeoperation sie dient.

4. Es muss bei einer Operation o1 mit mehreren Eingaben festgelegt werden, welche Ausgaben ihr zugeordnet sind und von welchen Vorgängeroperationen diese stammen.

5. Operationen können Parameter zugeordnet werden.

Um auch die eben aufgeführten Eigenschaften modellieren zu können muss die Struktur des gerichteten Graphen erweitert werden. Das Ergebnis dieser Erweiterung ist der *Algorithmengraph*. Im Hinblick der einfachen Implementierbarkeit soll die Definition des Algorithmengraphen objektorientiert erfolgen.

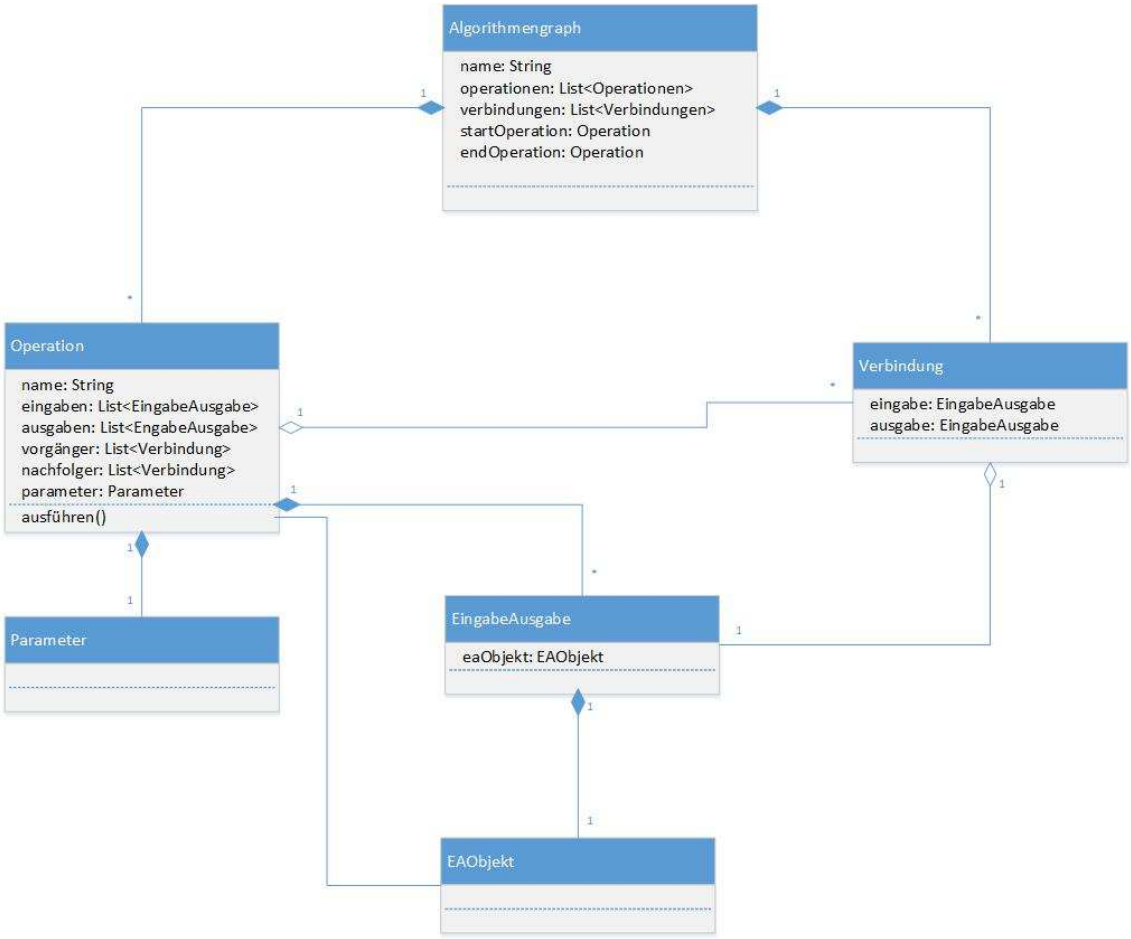


Abbildung 22: Klassenstruktur des Algorithmengraphen

Abbildung 22 gibt einen Überblick über die Klassen und deren Beziehung zur Modellierung eines Algorithmengraphen in der Form eines Klassendiagramms.

Eine Instanz der Klasse *Algorithmengraph* repräsentiert einen Ablauf der Segmentierung (Algorithmus). Sie hat ein Attribut *name* vom primitiven Typ *String*, dem ein sinnvoller Bezeichner des Algorithmus zugewiesen werden kann. Sie hält in ihrem *verbindungen*-Attribut und in ihrem *operationen*-Attribut eine Liste von *Verbindungs*- und *Operationen*-Objekten, die die Kanten und Knoten des Algorithmengraphen repräsentieren und von ihm existentiell abhängig sind. Deswegen bestehen zwischen der Klasse Algorithmengraph, der Klasse Verbindung und der Klasse Operationen Kompositionsbeziehungen. Der Typ Liste

ist hier als dynamische Datenstruktur, mit der miteinander in Beziehung stehende Objekte gespeichert werden können, zu verstehen. (vgl. [WIK16e])

Bis auf den Bezeichner entspricht die eben beschriebene Klassenstruktur der objektorientierten Beschreibung eines Graphen als abstrakte Struktur, die aus einer Menge von Knoten und einer Menge von Kanten besteht. Diese Klassenstruktur wird um die Attribute *startOperation* und *endOperation* erweitert. Beide sind Referenzen auf jeweils ein Element der operationen-Liste. Dem startOperation-Attribut wird die erste Operation, die auf das Eingabebild ausgeführt werden soll, zugewiesen. Dem endOperation-Attribut wird die letzte Operation des Segmentierungs-Algorithmus zugewiesen.

Eine Instanz der Klasse *Operation* repräsentiert einerseits einen Schritt (Operation) eines Segmentierungsalgorithmus und ist andererseits bezüglich des Algortihmengraphen als Knoten aufzufassen. Sie hat ein Attribut *name* vom Typ String, dem ein sinnvoller Bezeichner der Operation zugewiesen werden kann. Desweiteren verfügt sie über ein Attribut *eingaben* und ein Attribut *ausgaben*, die Objekte vom Typ *EingabeAusgabe* in einer Liste halten. EingabeAusgabe-Objekte sind existenziell abhängig von der jeweiligen Operation-Instanz, deswegen sind ihre Klassen mit einer Kompositions-Beziehung verknüpft. Ein EingabeAusgabe-Objekt ist ein Platzhalter für die Eingaben und Ausgaben einer Operation. Eine konkrete Eingabe oder Ausgabe kann ihnen in Form einer Instanz der Klasse *EAObjekt* als Wert ihres Attributs *eaObjekt* zugewiesen werden. Die Attribute *vorgänger* und *nachfolger* eines Operation-Objekts sind Listen von Instanzen des Typs *Verbindung*, über die sich die Beziehungen zu den Vor- bzw. Nachfolge-Operationen definieren lassen. Instanzen der Klasse Verbindung ordnen eine Ausgabe einer Operation einer Eingabe einer anderen Operation zu. Dies geschieht über das Zuweisen von Referenzen des jeweiligen Elements der *eingabe*- und *ausgabe*-Liste der beiden Operationen zu den Attributen *eingabe* und *ausgabe* der Verbindungs-Instanz. Dem *parameter*-Attribut der Klasse Operation kann ein Objekt vom Typ *Parameter* zugewiesen werden. Ein Parameter-Objekt enthält die Parameter zur Konfiguration der jeweiligen Operation. Die Methode ausführen der Operation-Klasse, führt die jeweilige Operation unter Verwendung der gegebenenfalls vorhanden Parameter aus. Dabei werden aus den eobjekte-Objekte (Eingabe-EObjekte) der EingabeAusgabe-Objeke des eingaben-Attributs eobjekte-Objekte (Ergebnis-EObjekte) erzeugt, die das Ergebnis der jeweiligen Operation repräsentieren. Diese werden dann den gegebenenfalls vorhandenen Nachfolgeoperationen übergeben. Dabei werden über eine Auswertung der ausgaben-Liste zu ermittelnde eingabe-Objekte der Nachfolgeknoten ermittelt, denen dann die Ergebnis-eobjekte zugeordnet werden. Die Endoperation hat allerdings keinen Nachfolgeknoten. Bei ihr werden die Ergebnis-EObjekte ihren Elementen der ausgaben-Liste zugeordnet.

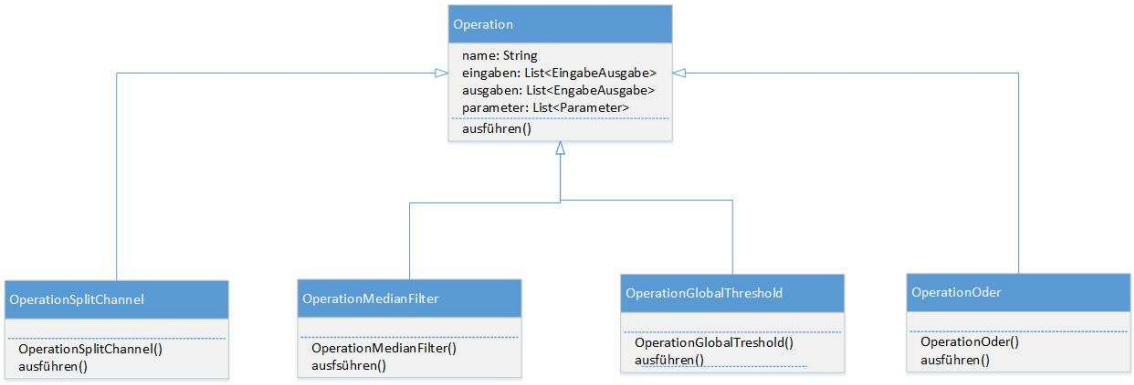


Abbildung 23: Beispiele für Operationen

Die Operationklasse, ist eine abstrakte Beschreibung von konkreten Bildbearbeitungs-Operationen. Abbildung 23 gibt einen Überblick über die Klassen von einigen konkreten Operationen und der Operationen-Klasse, sowie deren Beziehung. So ist eine konkrete Operation von der Operationen-Klasse abgeleitet. Sie überschreibt die Methode ausführen, so dass mit ihr der Segmentierungsschritt ausgeführt wird. In ihrem Konstruktor kann beispielsweise eine Zuordnung eines Bezeichners zu dem geerbten name-Attribut, sowie das Instanziieren von EingabeAusgabe-Objekten und deren Zuweisung zu den eingabe und ausgabe Attributen erfolgen.

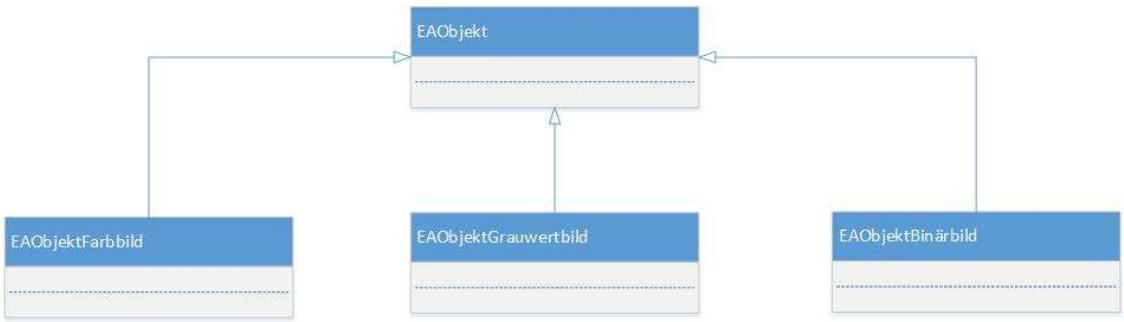


Abbildung 24: Beispiele für EAObjekt-Typen

In Abbildung 23 werden beispielhaft vier Klassen für konkrete Operationen aufgezeigt. Die Klasse *OperationSplitchannel* repräsentiert das im Anfang dieses Abschnitts aufgezeigte Extrahieren von Farbkanälen aus einem Farbbild. Die Klasse *OperationMedianFilter* repräsentiert einen Medianfilter, die Klasse *OperationGlobalThreshold* ein globales Schwellwertverfahren und die Klasse *OperationOder* eine Oder-Operation.

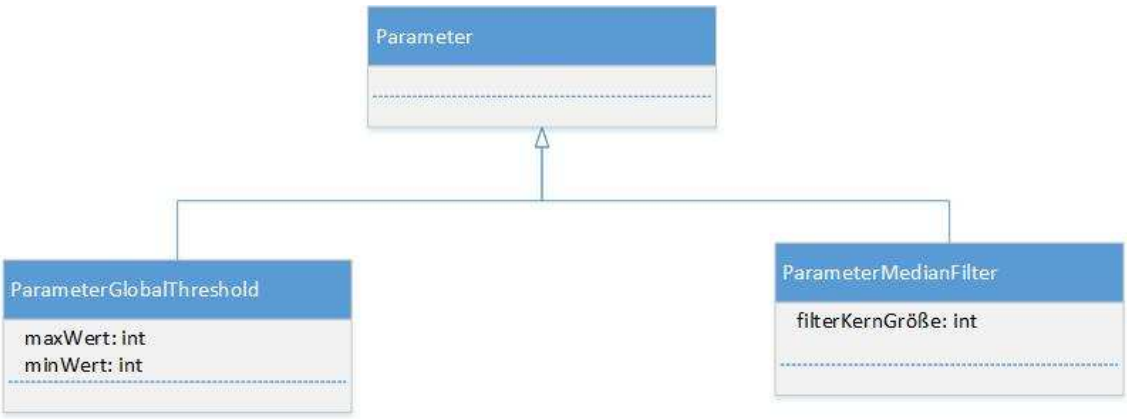


Abbildung 25: Beispiele für Parameter-Typen

Auch der Parametertyp ist eine Abstraktion. Seine konkreten Ausprägungen beinhalten die konkreten Parameter zur Konfiguration der jeweiligen Operation. Abbildung 25 zeigt die Beziehungen zwischen zwei Beispielen von konkreten Parameter-Klassen und der abstrakten Parameter-Klasse. Die Klasse ParameterGlobalThreshold definiert mit ihren Attributen *maxWert* und *minWert* die Min und Max Parameter des globalen Schwellwertverfahrens und die ParameterMedianFilter-Klasse mit ihrem Attribut *filterKernGröße* die Größe einer Filtermatrix eines Median-Filters. Beide erweitern die Parameterklasse.

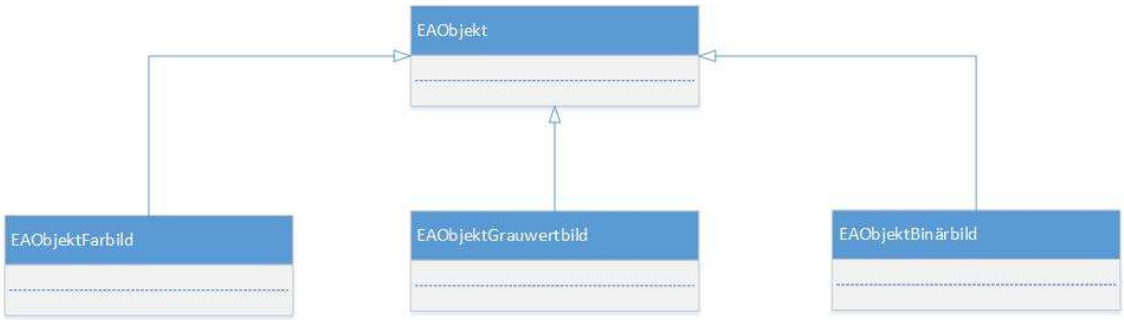


Abbildung 26: Beispiele für EAObjekt-Typen

Bildsegmentierungsoperationen werden auf verschiedenen Typen von Eingabebildern ausgeführt. So verlangt ein globales Schwellwertverfahren beispielsweise ein Grauwertbild als Eingabe, während eine Oder-Operation ein Binärbild als Eingabe hat. Ein Farbbild kann beispielsweise eine Eingabe für eine Splitchannel -Operation sein. Auch die Typen der Ergebnisbilder der jeweiligen Operationen lassen sich unterscheiden. So liefert ein globales Schwellwertverfahren und eine Oderoperation ein Binärbild, während eine Splitchannel-Operation mehrere Grauwertbilder als Ergebnis zur Folge hat. Diesem Umstand wird im Modell des Algorithmengraphen mit der Schaffung von Klassen des Typs EAObjekt Rechnung getragen. Abbildung 26 gibt einen Überblick über mögliche EAObjekt-Typen. So repräsentieren die Instanzen der Klasse EAObjektFarbild Farbbilder, die der Klasse EAObjektGrauwertbild Grauwertbilder und die der Klasse EAObjektbinärbild Binärbilder.

In den folgenden Ausführungen soll die Modellierung von Algorithmen mit dem eben vorgestellten Algorithmengraphen anhand eines Beispiels verdeutlicht werden.

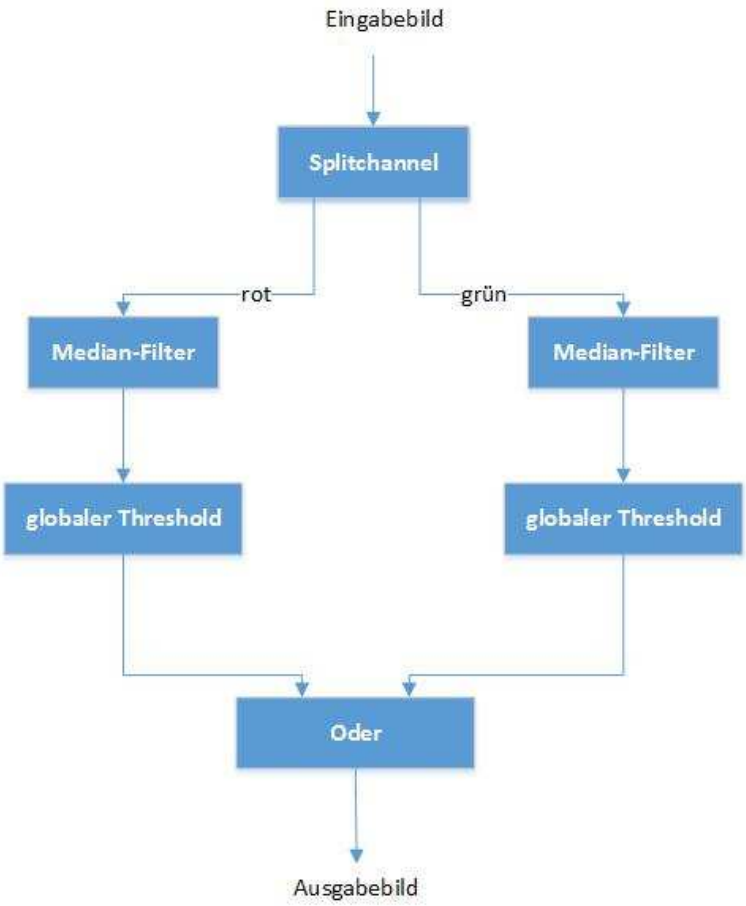


Abbildung 27: schematische Darstellung eines Segmentierungsalgorithmus

Abbildung 27 ist eine schematische Darstellung des Segmentierungsalgorithmus, der mit dem Algorithmengraphen modelliert werden soll. Das als Farbbild vorliegende Eingabebild wird zunächst mit einer Splitchannel-Operation in verschiedene Farbkanäle zerlegt. Auf dem rot und grün Kanal wird dann jeweils ein Median-Filter ausgeführt. Anschließend werden auf den Ergebnisbildern der Median-Filter globale Schwellwertverfahren angewendet. Die Ergebnisse der Schwellwert-Operationen werden abschließend mit einer Oder-Operation verknüpft.

- Für das Modellieren dieses Segmentierungsvorgangs ist als erstes eine Instanz der Klasse Algorithmengraph anzulegen und ihr einen Bezeichner zuzuordnen. Dieser Bezeichner kann beispielsweise aus den Anfangsbuchstaben der Operationen zusammengestzt werden "SCMFGTGTO".
- Nun werden Objekte der Operationen instanziiert. Dabei wird ein Objekt der SplitchannelOperation, zwei Objekte vom Typ OperationMedianFilter, zwei Objekte der OperationGlobalThreshold-Klasse und ein Objekt der OperationOder erzeugt. Diese Operation-Objekte werden der operationen-Liste des Algorithmengraph-Objekts hinzugefügt. Dann wird auf das startOperation-Attribut und auf das endOperationen-Attribut des Algorithmengraph-Objekts eine Referenz der Instanz der OperationSplitchannel und der Instanz der OperationOder gesetzt.

- Jetzt werden sechs Objekte der Klasse Verbindung-Objekte instanziiert und der verbindungen-Liste des Algorithmengraphen hinzugefügt.
- Zwei Verbindungen werden so konfiguriert, dass die EingabeAusgabe-Objekte der Rot- und der Grün-Ausgaben der SplitchannelOperation den Eingängen der beiden OperationMedianFilter-Objekten zugeordnet werden.
- Analog erfolgt abschließend die Konfiguration der Verbindungen der beiden OperationMedianFilter-Objekte mit jeweils einem OperationGlobalThreshold-Objekt und die Konfiguration der Verbindungen der beiden OperationGlobalTreshorld-Objekte mit dem OperationOder-Objekt.

Mit dem in diesem Abschnitt vorgestellten Modell des Algorithmen-Graphen lassen sich Algorithmengraphen definieren, die formal zwar korrekt, aber semantisch widersprüchlich und fehlerhaft sind. So ist es beispielsweise möglich Zyklen zu erzeugen, in denen eine Operation mit sich selbst verbunden wird. Desweiteren können auch Operationen miteinander verbunden werden, die bezüglich ihrer Hintereinanderausführung inkompatibel sind. Es bleibt dem Anwender überlassen auf die semantische Korrektheit der Algorithmengraphen zu achten.

4.3 Analysegraph als Modell für die Analyse und Konfiguration von Algorithmen

Nachdem im vorangegangenen Abschnitt 4.2 der Algorithmengraph als Modell für Segmentierungsalgorithmen vorgestellt wurde, soll nun dieses Modell um Informationen zur Analyse und Konfiguration von Algorithmen erweitert werden. Unter Konfiguration ist hier das Erstellen von Parameter-Objekten und deren Zuweisung zu den Operationen eines Algorithmengraphen zu verstehen.

Definition 5. Konfiguration eines Algorithmengraphen
Die Konfiguration eines Algorithmengraphen ist die Belegung aller Parameter seiner Operationen.

Definition 6. Konfigurationseigenschaft eines Algorithmengraphen
Ein Algorithmengraph heißt konfiguriert, wenn seinen parametrisierbaren Operationen Parameterobjekte mit Belegungen zugeordnet sind.

Unter Analyse eines Algorithmengraphen ist das Bestimmen des kleinstmöglichen Segmentierungsfehlers für eine Zielklasse, des aus Anwendung des Algorithmengraphen resultierenden Ergebnisbildes, zu verstehen. (siehe Abschnitt 4.1) Die Voraussetzung für die Analyse eines Algorithmengraphen ist das Finden einer möglichst optimalen Konfiguration seiner Operationen, in denen der Segmentierungsfehler minimiert wird.

Die möglichen Werte der Parameter aller Operationen lassen sich als mathematischer Raum (Parameter-Raum) auffassen. Eine Stelle in diesem Raum ist eine Konfiguration des Algorithmengraphen. Für das Finden einer optimalen Konfiguration gilt es den Parameterraum nach Stellen zu durchsuchen, in denen der Segmentierungsfehler minimal ist. Eine Methode dies zu tun, wäre beispielsweise die Brute-Force-Methode, also das Ausprobieren aller möglichen Kombinationen von Parameterbelegungen. Bei komplexeren Algorithmengraphen würde man so zwar mit Sicherheit eine optimale Belegung finden, aber der

Rechenaufwand würde schnell überaus groß werden. Deswegen wurde sich in dieser Arbeit für eine heuristische Herangehensweise für die Ermittlung von optimalen Konfigurationen entschieden.

Die grundlegende Idee zur Analyse und Konfiguration von Algorithmengraphen ist es, die Operationenknoten sukzessive abzuarbeiten und für sie eine lokale optimale Konfiguration zu finden. Für diese Vorgehensweise wird Wissen darüber benötigt, wie eine Operation analysiert werden kann. Dieses Wissen kann mit einem Algorithmengraphen nicht dargestellt werden. Für die Modellierung dieses Wissens muss der Algorithmengraph erweitert werden. Das Ergebnis dieser Erweiterung ist der Analysegraph. Doch bevor der Analysegraph vorgestellt wird, soll die objektorientierte Modellierung des im Abschnitt 4.1 vorgestellten A-Priori Wissens des SM-Verfahrens und des Segmentierungsfehlers erfolgen.

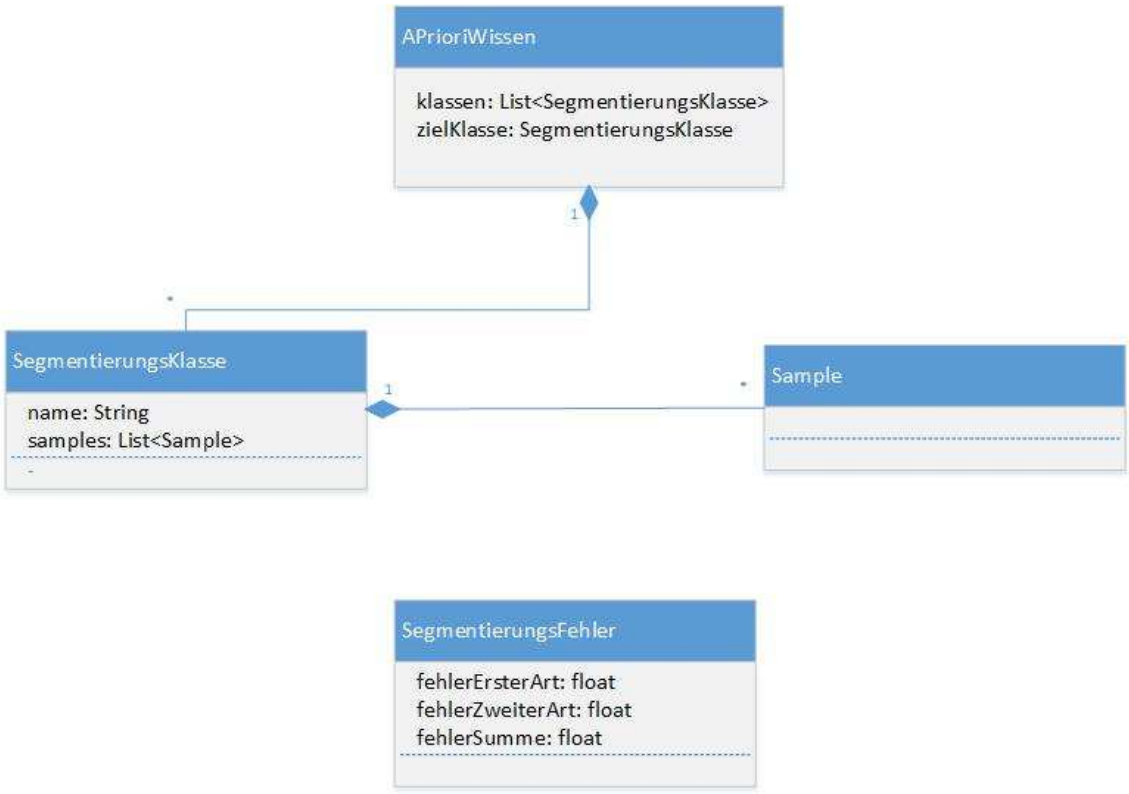


Abbildung 28: Klassendiagramm A-Priori Wissen des SM-Verfahren und Segmentierungsfehler

Abbildung 28 verdeutlicht die Klassenstruktur des A-Priori Wissens und des Segmentierungsfehlers. Eine Instanz der Klasse *AprioriWissen* repräsentiert das A-Priori Wissen für ein SM-Verfahren. Sie hat ein Attribut *klassen* vom Typ Liste, deren Elemente alle Segmentierungsklassen-Objekte sind. Das Attribut *zielklasse* referenziert ein Segmentierungsklassen-Objekt aus der klassen-Liste, wodurch die Zielklasse für das SM-Verfahren festgelegt wird. Ein Objekt der Klasse *Segmentierungsklasse* repräsentiert eine der im Abschnitt 4.1 definierten Klasse. Sie hat ein Attribut *name* vom Typ String, dessen Wert der Bezeichner der Klasse ist. In ihrer Liste *samples* werden Objekte der *Sampleklasse* gehalten, die die ihr zugeordneten Stellen im Bild (Bildpunkte) definieren. Die Klasse *Segmentierungsfehler* beinhaltet den im Abschnitt 4.1 definierten Segmentierungsfehler.

Nach dem eben das objektorientierte Modell des A-Priori Wissens vorgestellt wurde, soll nun die Klassen-

struktur des Algorithmengraphen erläutert werden.

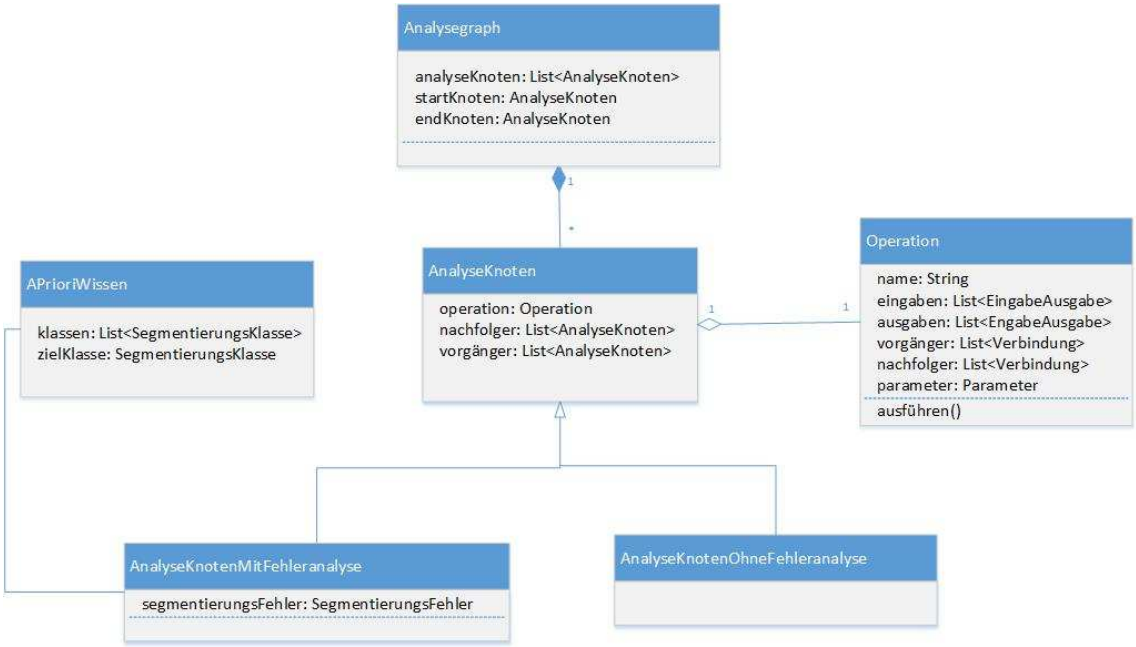


Abbildung 29: Klassendiagramm Analysegraph

Abbildung 29 gibt einen Überblick über die Klassen des Analysegraphen. Eine Instanz der Klasse *Analysegraph* repräsentiert einen Analysegraphen. Sie hält in ihrem Attribut *analyseKnoten* eine Liste von *AnalyseKnoten*-Objekten. Ihre Attribute startKnoten und endKnoten definieren durch Referenzen auf AnalyseKnoten-Objekte den Start- und Endpunkt des Algorithmengraphen. Ein *AnalyseKnoten*-Objekt repräsentiert einen Schritt des Analyseverfahrens. Seinem Attribut *operation* ist eine Operation des Algorithmengraphen zugeordnet, die analysiert werden soll. Die Attribute *vorgänger* und *nachfolger* definieren Vorgänger- und Nachfolger-Analyseknoten.

Es gibt bezüglich der Fehleranalyse Eigenschaften von Operationen, aus denen sich verschiedene Typen von AnalyseKnoten ableiten lassen. Tabelle 1 gibt eine Übersicht über diese Eigenschaften.

mit Fehleranalyse		ohne Fehleranalyse	
mit Parameter	ohne Parameter	mit Parameter	ohne Parameter

Tabelle 1: Übersicht über die Eigenschaften von Operationen bezüglich ihrer Analyse

So lassen sich Operationen prinzipiell darin unterscheiden, ob man bei ihnen eine lokale Fehleranalyse ausführen kann oder nicht. So macht eine Fehleranalyse beispielsweise bei einer MedianFilter-Operation keinen Sinn, da diese noch kein segmentiertes Binärbild als Ergebnis hat. Bei einer globalen Schwellwert-Operation kann eine Fehleranalyse erfolgen, weil sie ein segmentiertes Binärbild zur Folge hat, für das die Bestimmung des Segmentierungsfehlers erfolgen kann. Ein weiteres Kriterium, anhand dessen man Operationen unterscheiden kann, ist, ob eine Operation Parameter hat oder nicht. So hat eine binäre Oder-Operation beispielsweise keinen Parameter, während eine globale Threshold-Operation als Parameter

die Min und Max Werte hat. Aus der Kombination dieser zwei Unterscheidungsmerkmale lassen sich Operationen in vier Kategorien einteilen:

1. Operationen mit Fehleranalyse und mit Parameter
2. Operationen mit Fehleranalyse und ohne Parameter
3. Operationen ohne Fehleranalyse und mit Parameter
4. Operationen ohne Fehleranalyse und ohne Parameter

Aus diesen vier Kategorien von Operationen leiten sich vier verschiedene Typen von Analyseknoten ab, die mit der nun folgenden Erläuterung der Abbildungen 30 und 32 vorgestellt werden.

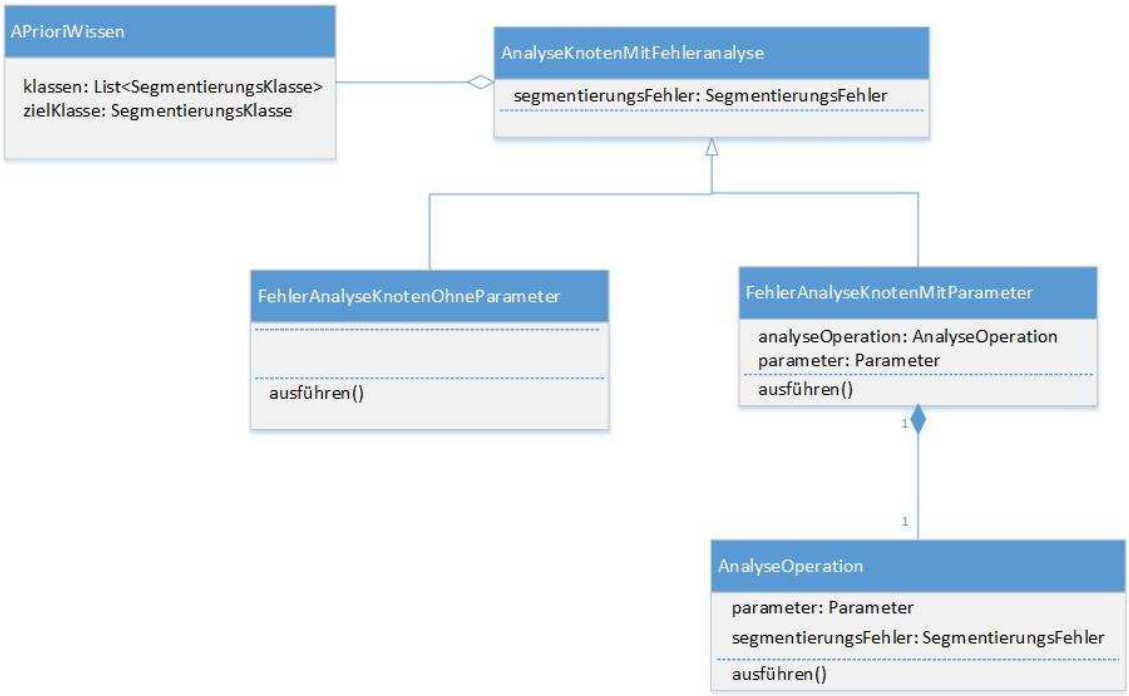


Abbildung 30: AnalyseKnotenMitFehleranalyse

Abbildung 30 gibt einen Überblick über die Analyseknoten mit Fehleranalyse. Ihre Subklasse *AnalyseKnotenMitFehleranalyse* erweitert die Klasse Analyseknoten. Ein AnalyseknotenMitFehleranalyse ist meinem AprioriWissen-Objekt assoziiert, anhand dessen die Fehleranayse durchgeführt werden soll. Desweiteren hat es ein Attribut segmentierungsFehler vom Typ SegmentierungsFehler, dem das Ergebnis der Fehleranalyse als Wert zugeordnet wird. Die beiden Klassen *FehlerAnalyseKnotenOhneParameter* und *FehlerAnalyseKnotenOhneParameter* erweitern die Klasse AnalyseKnotenMitFehlerAnalyse. Die Methode ausführen FehlerAnalyseKnotenOhneParameter führt die ihr zugeordnete Operation des Algorithmenknotens aus und ermittelt auf deren Ergebnis gemäß der in der Definition 3 im Abschnitt 4.1 vorgestellten Berechnung den Segmentierungsfehler und ordnet diesen ihrem geerbten Attribut segmentierungsFehler

zu. Bei einem FehlerAnalyseKnotenMitParameter-Objekt müssen für die ihm zugeordnete Operation geeignete Parameter ermittelt werden. Da das hierfür durchgeführte Verfahren spezifisch zu der jeweiligen Operation ist, wird dieses mit einer weiteren Klasse *AnalyseOperation* abstrahiert. Eine Instanz der Klasse AnalyseOperation ermittelt mit ihrer Methode *ausführen* die für die Operation optimalen Parameter und den minimalen Segmentierungsfehler. Die ausführen-Methode der AnalyseOperation wird von der *ausführen*-Methode der Klasse FehlerAnalyseKnotenMitParameter aufgerufen. Diese ordnet darüberhinaus auch die ermittelten optimalen Parameter der jeweiligen Algorithmengraph-Operation zu.

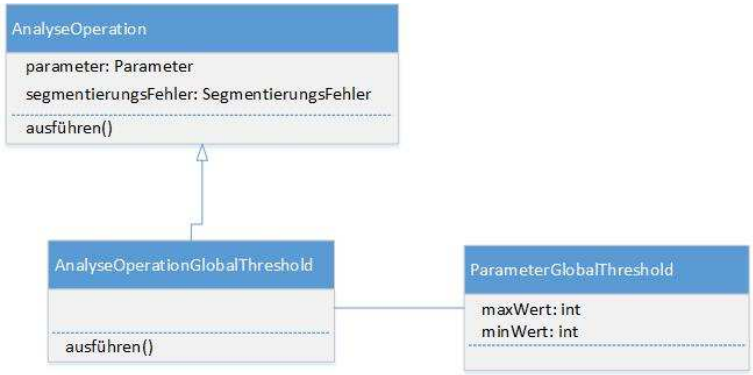


Abbildung 31: Beispiel für AnalyseOperation

Abbildung 31 zeigt ein Beispiel für eine konkrete AnalyseOperation. Die Klasse *AnalyseOperationGlobalThreshold* erweitert die Klasse AnalyseOperation und überschreibt deren ausführen-Methode, so dass mit ihr die optimalen Parameter eines globalen Schwellwertverfahrens also ein ParameterGlobalThreshold-Objekt ermittelt werden. Das Ermitteln der Parameter kann beispielsweise per Brutforce geschehen, in dem die GlobalThreshold-Operation auf deren Eingabebild mit allen sinnvollen Kombinationen von minWert und maxWert, bei denen gilt: minWert<maxWert, angewendet wird und dessen Ergebnisse mit Hilfe des AprioriWissen bezüglich des Segmentierungsfehlers bewertet werden.

Die Klassen der AnalyseKnoten ohne Fehleranalyse sind in Abbildung 32 dargestellt. Ihre Subklasse *AnalyseKnotenOhneFehleranalyse* erweitert die Klasse AnalyseKnoten. Von AnalyseKnotenOhneFehleranalyse sind die Klassen *AnalyseKnotenMitParameter* und *AnalyseKnotenOhneParameter* abgeleitet. Eine Instanz der Klasse AnalyseKnotenOhneParameter kennzeichnet lediglich die ihr zugeordnete Operation des Algorithmengraphen als fehler- und parameterlos. Eine Operation. die der Instanz der Klasse AnalyseKnotenMitParameter zugeordnet ist, hat zwar Parameter aber eignet sich nicht für eine Fehleranalyse. Die Untersuchung des Parameterraums erfolgt daher in der Abhängigkeit eines Nachfolge-Analyseknotens mit Fehleranalyse. Die Definition des AnalyseKnotenMitFehleranalyse erfolgt durch dessen Referenzierung über das Attribut *abhängigerKnoten* der AnalyseKnotenMitParameter-Klasse.

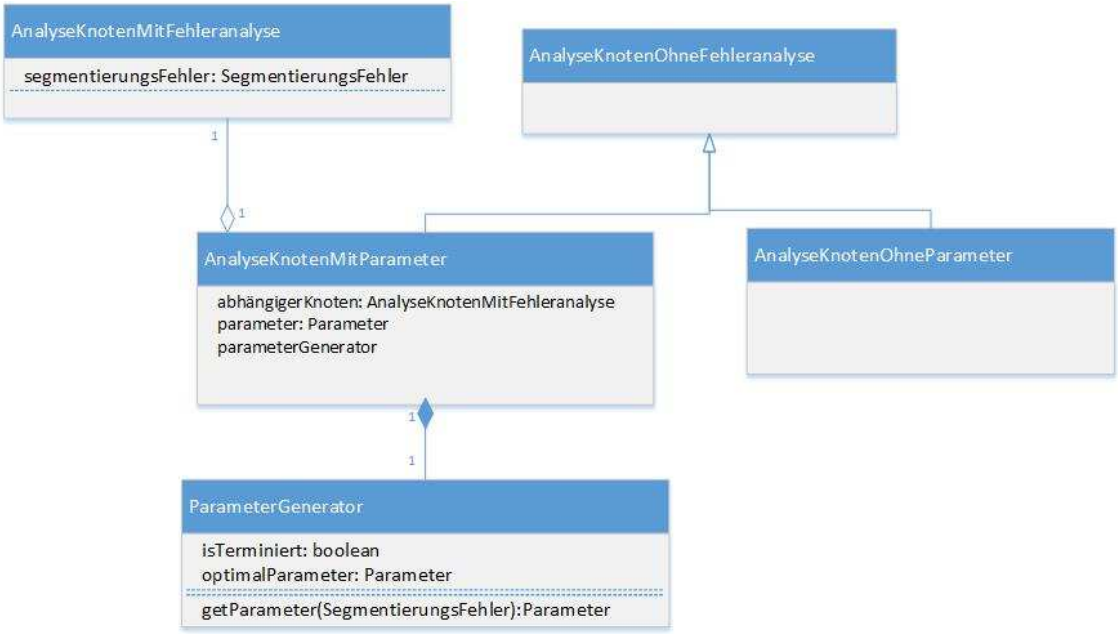


Abbildung 32: AnalyseKnotenOhneFehleranalyse

Der AnalyseKnotenMitFehleranalyse liefert SegmentierungsFehler-Objekte mit denen die Parameter bewertet werden können. Diese Bewertung geschieht mit Instanzen der Klasse *ParameterGenerator*. Ein ParameterGenerator-Objekt generiert mit seiner Methode *getParameter* Parameter für die Algorithmengraph-Operation. Ihr wird der Segmentierungsfehler des AnalyseKnotenMitFehleranalyse übergeben, anhand dessen sie entscheiden kann, ob noch weitere Parameter generiert werden sollen oder nicht und welcher der bisher generierten Parameter optimal ist. Sie modifiziert dafür die Attribute *isTerminiert* und *optimalParameter* des ParameterGenerator-Objekts.

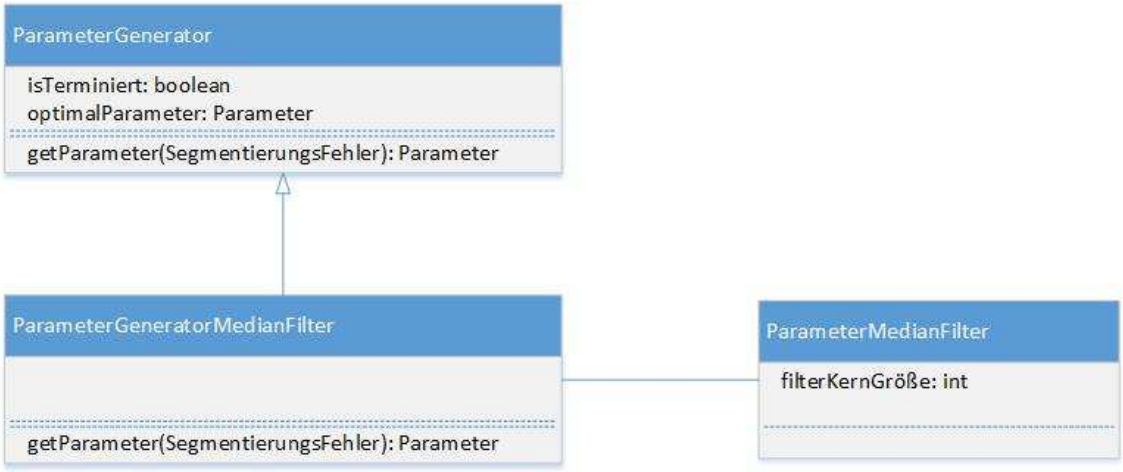


Abbildung 33: Beispiel für ParameterGenerator

Die Funktionsweise des ParameterGenerators ist abhängig von der Operation des Algorithmengraphens, die dem AnalyseKnotenMitParameter zugeordnet ist. Daher kann sie verschiedene Ausprägungen haben. Abbildung 33 zeigt ein Beispiel für einen konkreten ParameterGenerator. Die Klasse *ParameterGeneratorMedianFilter* erweitert die Klasse ParameterGenerator und überschreibt deren getParameter-Methode, so dass sie ein ParameterMedianFilter-Objekt für eine MedianFilter-Operation zurückgeben wird. Eine Implementierung der getParameter-Methode könnte beispielsweise eine begrenzte Anzahl verschiedener Größen von Filtermatrizen zurückgeben und dann terminieren, wenn der Segmentierungsfehler größer wird oder alle möglichen Werte der Filtermatrixwerte zurückgegeben wurden.

Nachdem in den bisherigen Ausführungen dieses Abschnitts die Beschreibung des Analysegraphen erfolgte, soll nun abschließend desse Anwendung zur Modellierung von Algorithmenanalysen anhand eines Beispieles verdeutlicht werden. Dabei wird das Beispiel am Ende des Abschnitts 4.2 aufgegriffen.

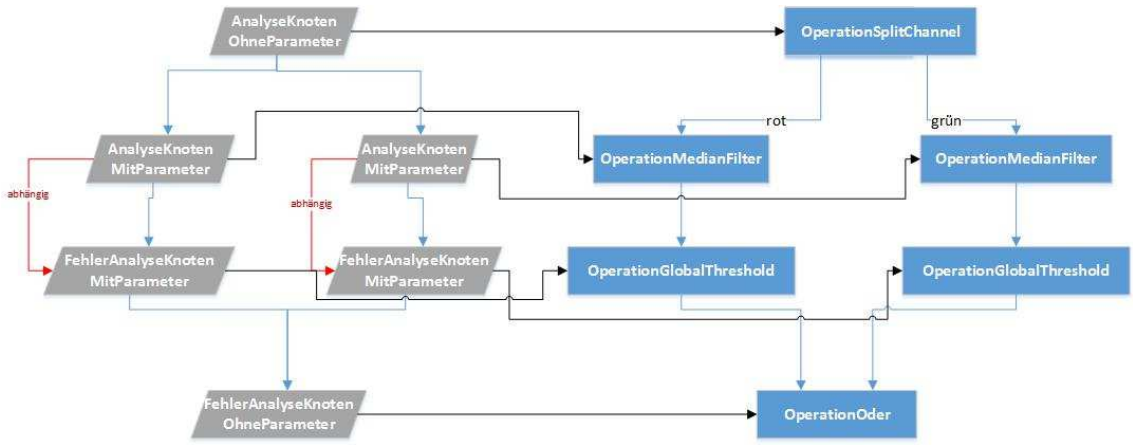


Abbildung 34: Schematische Darstellung von einem Algorithmengraph und dem zugehörigen Analysegraph

Abbildung 34 gibt einen schematischen Überblick über den in Abschnitt 4.2 vorgestellten Algorithmengraphen und den Analysegraphen mit dem er erweitert werden soll.

Im Allgemeinen ist für die Erstellung eines Analysegraphen zunächst ein AnalyseGraph-Objekt zu instanziieren. Für jede Operation des Algorithmengraphen ist ein Analyseknoten anzulegen, der die jeweilige Operation referenziert. Der Typ eines Analyseknotes hängt dabei von den Eigenschaften der jeweiligen Operation ab. Nun sind die Start- und End-Analyseknoten zu definieren. Dies geschieht durch Zuweisen der Referenzen zu den startKnoten- und endKnoten-Attributen des AnalyseGraphen-Objekts. Im Anschluss daran werden die Vorgänger- Nachfolger-Beziehungen der Analyseknoten durch Einfügen der entsprechenden Referenzen in die vorgänger- nachfolger-Listen der Analyseknoten definiert. Für FehleranalyseKnotenMitParameter sind Operations-spezifische AnalyseOperation-Objekte zu instanziieren und dem analyseOperation-Attribut des jeweiligen FehleranalyseKnotenMitParameter-Objekts zuzuweisen. Für AnalyseKnotenMitParameter sind die AnalyseKnotenMitFehleranalyse, von denen sie abhängen zu referenzieren und Operations-spezifische ParameterGenerator-Objekte zu instanziieren, die dem parameterGenerator-Attribut des jeweiligen AnalyseKnotenMitFehleranalyse zugewiesen werden.

Der eben dargestellte Ablauf der Modellierung eines Analysegraphen gestaltet sich angewandt auf das vorliegende Beispiel wie folgt:

- Zuerst wird ein Analysegraph-Objekt instanziiert.
- Dann erfolgt die Instanzierung der Analyseknoten. So wird ein AnalyseKnotenOhneParameter-Objekt für die OperationSplitchannel erzeugt und das OperationSplitchannel-Objekt wird dem operation-Attribut des AnalyseKnotenOhneParameter-Objekt zugewiesen. Für die Beiden OperationMeanFilter-Objekte wird jeweils ein AnalyseKnotenMitParameter instanziiert, denen die OperationMeanFilter-Objekte zugewiesen werden. Für die beiden OperationGlobalThreshold-Objekte werden zwei FehleranalyseKnotenMitParameter-Objekte instanziiert, denen die OperationGlobalThreshold-Objekte zugewiesen werden. Für die OperationOder wird ein FehleranalyseKnotenOhneParameter erzeugt, dem das OperationOder-Objekt zugewiesen wird.
- Dem startKnoten-Attribut des AnalyseGraph-Objekts wird der AnalyseKnotenOhneParameter-Objekt der OperationSplitchannel und dem endKnoten-Attribut das FehleranalyseKnotenOhneParameter-Objekt zugewiesen.
- Nun erfolgt die Definition der Vorgänger- Nachfolger-Beziehungen. Dabei werden in der nachfolger-Liste des AnalyseKnotenOhneParameter-Objekts der OperationSplitchannel die beiden AnalyseKnotenMitParameter der OperationMeanFilter-Objekte eingefügt. In die vorgänger-Listen der AnalyseKnotenMitParameter-Objekte der OperationMeanFilter-Objekte wird das AnalyseKnotenOhneParameter-Objekt der OperationSplitchannel eingefügt. Den nachfolger-Listen der AnalyseKnotenMitParameter-Objekte werden jeweils eines der beiden FehleranalyseKnotenMitParameter-Objekte der OperationGlobalThreshold-Objekte zugeordnet. Den beiden nachfolger-Liste der FehleranalyseKnotenMitParameter-Objekte der OperationGlobalThreshold-Objekte wird der FehleranalyseKnotenOhneParameter der OperationOder zugewiesen und der vorgänger-Liste des FehleranalyseKnotenOhneParameter-Objekts werden die beiden FehleranalyseKnotenMitParameter-Objekte zugewiesen.

- Den abhängigerKnoten-Attributen der beiden AnalyseKnotenMitParameter-Objekte wird jeweils eines der FehleranalyseKnotenMitParameter-Objekte, das auch in der nachfolge-Liste des jeweiligen AnalyseKnotenOhneParameter-Objekts enthalten ist, zugeordnet.
- In Anschluss daran werden zwei AnalyseGlobalThreshold-Objekte instanziiert und den analyseOperationen-Attributen der FehleranalyseKnotenMitParameter-Objekte zugeordnet.
- Abschließend werden zwei ParameterGeneratorMedianFilter-Objekte instanziiert und den parameterGenerator-Attributen der beiden AnalyseKnotenMitParameter zugewiesen.

4.4 Ausführungsalgorithmen für Analysegraph und Algorithmengraph

Nachdem in den vorangegangen Abschnitten 4.2 und 4.3 mit dem Analysegraph und dem Analysegraph Modelle für Bildsegmentierungsalgorithmen und deren Analyse vorgestellt wurden, soll nun Algorithmen zur Interpretation der beiden Graphen vorgestellt werden.

Der Algorithmus zur Ausführung eines Algorithmengraphen erzeugt aus einen farbigen Eingabebild durch Interpretieren eines konfigurierten Algorithmengraphen und ein segmentiertes Binärbild. Er wird wie folgt definiert:

Eingabe: konfigurierter Algorithmengraph-Objekt, Eingabebild in Form eines EAObjektes

Ausgabe: Binärbild in Form eines EAObjektBinärbild

Schritt 1: Weise dem EngabeAusgabe-Objekt, das sich in der eingabe-Liste der Start-Operation des Algorithmengraphen befindet, das Eingabe-EAObjektFarbbild zu.

Schritt 2: Rufe die Funktion operationAusführen aus, übergebe ihr dabei als Parameter die End-Operation des Algorithmengraphen.

Schritt 3: Das eaObjekt-Attribut des EingabeAusgabe-Objekts, das mit der ausgabe-Liste der End-Operation referenziert ist, hat als Wert ein EAObjektBinärbild. Gib dieses EAObjektBinärbild als Ergebnis zurück.

operationAusführen:

```
1  operationAusführen(Operation op){
2   Wenn die ausführen-Methoder von op schon aufgerufen wurde dann{
3    return;
4   }
5    Wenn op die Start-Operation ist dann{
6    Führe  op.ausführen() aus;
7    Merke dir, dass op ausgeführt wurde;
8    return;
9   }
10  Wenn op vorgänger-Verbindung hat dann{
11   Führe für jede Operation opv, die mit dem ausgabe-Attribut einer der
        vorgänger-Verbindung referenziert  wurde   OperationAusführen(opv)
        aus;
12 }
13  Führe  op.ausführen() aus;
14   Merke dir, dass op ausgeführt wurde;
15  return;
16 }
```

Der eben beschriebene Algorithmus funktioniert für semantisch korrekte Algorithmengraphen. So darf der Algorithmengraph beispielsweise keine Zyklen enthalten, seine Operationen müssen miteinander transitiv über Vorgänger- Nachfolger-Beziehungen miteinander verbunden sein und Operationen die in einer direkten Vorgänger- Nachfolger-Beziehung verknüpft sind, müssen zueinander kompatibel sein. Gegebenenfalls müsste vor Anwendung des beschriebenen Algorithmus noch eine Prüfung auf die semantische Korrektheit des Algorithmengraphen erfolgen. Der Umgang mit semantisch inkorrekten Algorithmengraphen soll jedoch nicht Gegenstand dieser Arbeit sein.

Der Algorithmus zur Ausführung eines Analysegraphen ermittelt die optimale Konfiguration eines Algorithmengraphen für dessen Anwendung auf einem Eingabebild bezüglich einer Zielklasse und den dazugehörigen Segmentierungsfehler. Er ist folgendermaßen definiert:

Eingabe: ein Analysegraph-Objekt und den dazugehörigen Algorithmengraph, ein Eingabebild in Form eines EAObjekts und ein AprioriWissen-Objekts

Ausgabe: ein konfigurierter AnalyseKnoten und ein SegmentierungsFehler-Objekt

Schritt 1: Weise dem EngabeAusgabe-Objekt, das sich in der eingabe-Liste der Start-Operation des Algorithmengraphen befindet, das Eingabe-EAObjektFarbbild zu.

Schritt 2: Führe die Funktion analysiereAnalyseKnoten aus, übergebe ihr dabei als Parameter den mit dem startKnoten-Attribut des AnalyseGraphen-Objekts referenzierten AnalyseKnoten.

Schritt 2: Gib den nun konfigurierten AnalyseGraphen. Gib das SegmentierungsFehler-Objekt aus. Dieser wird dem segmentierungsFehler-Attribut des AnalyseknotenMitFehleranlyse referenziert, der wiederum mit dem endKnoten-Attribut referenziert ist.

analysiereAnalyseKnoten:

```
1  analysiereAnalyseKnoten(AnalyseKnoten ak){
2   Wenn es eine Eingabe von ak.operation gibt, der kein EAObjekt
       zugewiesen wurde dann {
3     return;
4   }
5    Wenn ak vom Typ AnalyseKnotenMitFehleranalyse ist{
6      Führe ak.ausführen() aus;
7      Wenn ak ein als abhängig markierter Knoten ist{
8        return;
9      }
10   }
11   Wenn ak vom Typ AnalyseKnotenMitParameter ist{
12   Instanziiere einen SegmentierungsFehler-Objekt  fehler;
13   Setze fehler.fehlerErsterArt=1;fehler.ZweiterArt=1;fehler.fehlerSumme
       =2;
14
15   Solange ak.parameterGenerator.isTerminate=false gilt{
16
17       Kopiere den AnalyseGraph und den AlgorithmenGraph im aktuellen
           Zustand;
18       Bestimme in der Kopie des AnalyseGraphens den AnalyseKnoten akKopie
            der sich an der
19       selben Stelle wie ak befindet;
20         Führe Parameter p = ak.parameterGenerator.getParameter(fehler)
              aus;
21       Setze akKopie.operation.parameter=p;
22         führe akKopie.operation.ausführen() aus;
23       Für  alle akKopie.nachfolger aknf{
24         führe analysiereAnalyseKnoten(aknf) aus;
25       }
26      }
27    setze ak.operation = ak.parameterGenerator.optimalParameter
28   }
29   führe ak.operation.ausführen() aus;
30   Für  alle ak.nachfolger aknf{
31    führe analysiereAnalyseKnoten(aknf) aus;
32    }
33 }
```

5 Implementierung eines Prototyps

5.1 Verwendete Technologien

5.1.1 Überblick

In den folgenden Abschnitten sollen zunächst die für die Implementierung des Prototyps verwendeten Technologien kurz vorgestellt werden. Der Prototyp wird als *Eclipse Rich Client Platform* (RCP)-Anwendung umgesetzt. (Abschnitt 5.1.2) Für die Implementierung des AlgorithmenGraphen und des AnalyseGraphen sowie deren Persistenz wird das Eclipse Modeling Framework (EMF) eingesetzt. (Abschnitt 5.1.3)

5.1.2 Die Eclipse Rich Client Platform

Ab der Version 3.0 ist Eclipse nicht nur als integrierte Entwicklungsumgebung (IDE) sondern auch als Plattform für Desktopanwendungen interessant. (vgl. [WB16]) Denn ab dieser Version sind ihre Komponenten soweit entkoppelt, dass es möglich ist, allgemein benutzbare Komponenten ohne die IDE spezifischen Komponenten zu verwenden. Die allgemein nutzbaren Komponenten wurden in der *Eclipse Rich Client Platform* (RCP) zusammengefasst. (vgl. [WB16]) "Eclipse RCP (Rich Client Platform) ist eine Plattform zur Entwicklung von Desktop- Anwendungen. Sie stellt ein Grundgerüst bereit, das um eigene Anwendungsfunktionalitäten erweitert werden kann." ([Ebe11] S. 2 Z. 1 ff.) Zur Entwicklung von RCP Anwendungen stellt Eclipse eigene IDE-Komponenten bereit. Diese werden in einer auf der Eclipse-Download-Seite angebotenen Eclipse Version "Eclipse for RCP and RAP Developers" mitgeliefert. (vgl. [EDO16])

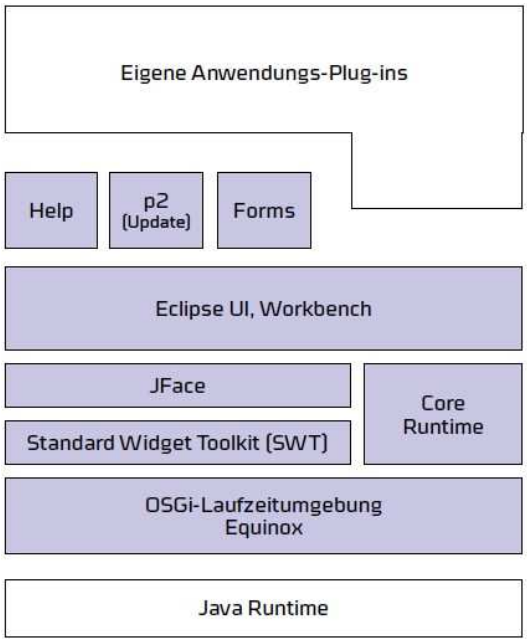


Abbildung 35: Übersicht über die Komponenten der Eclipse RCP ([Ebe11] S. 3)

Abbildung 35 gibt einen Überblick über die Komponenten der Eclipse RCP. Aus ihr geht hervor, dass die RCP auf der Java Runtime aufsetzt. Einige Komponenten sollen an dieser Stelle kurz vorgestellt werden.
OSGi-Laufzeitumgebung Equinox: Open Services Gateway initiative (OSGi) ist eine Spezifikation eines auf der Java Plattform basierenden Frameworks für die Entwicklung modularer Anwendungen. (vgl. [Ebe11] S. 60) OSGi wird von einem Industriekonsortium namens OSGi Alliance entwickelt und veröffentlicht. (vgl. [Ebe11] S. 60) Equinox implementiert die Kernspezifikationen von OSGi und erweitert diese.
Core Runtime: Eclipse Core Runtime stellt Funktionalitäten, wie das Verwalten von Lebenszyklen von Eclipse Anwendungen bereit. (vgl. [Ebe11] S. 3)

SWT und JFace: Das Standard Widget Toolkit (SWT) und JFace bieten Komponenten für die Entwicklung von graphischen Benutzeroberflächen.

Eclipse UI, Workbench: Das Eclipse User Interface (Eclipse UI) stellt mit seiner Workbench (deutsch Arbeitsbank) eine Art Rahmen- und Verwaltungsstruktur für GUI-Komponenten von Plug-ins bereit. Eine RCP-Anwendung kann mit ihrer Hilfe auf die von der Eclipse IDE bereits verwendeten Bedienkonzepte, wie beispielsweise View- und Editor-Reiter, Perspektiven und Menüstrukturen zurückgreifen. (vgl. [Ebe11] S. 4)
"Die Workbench macht die graphische Benutzeroberfläche einer RCP-Anwendung aus. Der grundlegende Aufbau der Benutzeroberfläche ist dabei durch Eclipse RCP vorgegeben und ist durch die Anwendung lediglich mit Inhalten zu befüllen." ([Ebe11] S. 18 Zeile 1-3)

Equinox spezifiziert Anwendungen als eine Menge von Modulen (Bundles oder Plug-ins genannt). (vgl. [Ebe11] S. 60) Diese Plug-ins werden innerhalb der Equinox Laufzeitumgebung ausgeführt. Sie besitzen einen Lebenszyklus, der von Equinox gesteuert wird. So können sie während der Laufzeit installiert, aktiviert, gestoppt und deinstalliert werden. (vgl. [Ebe11] S. 60)

Plug-ins können mit Hilfe der Eclipse IDE entwickelt werden. Sie werden hierbei durch sogenannte Plug-in-Projekte repräsentiert. Plug-in-Projekte bestehen im Wesentlichen aus klassischen Java-Programmelementen, Ressourcen, einer Extensible Markup Language- (XML-) Datei namens *plugin.xml* und einer *Manifest-Datei*. (vgl. [Ebe11] S. 60). Die Manifest-Datei enthält Metadaten des Plug-ins, die von Equinox für das Ausführen des Plug-ins benötigt werden. (vgl. [Ebe11] S. 62) So kann beispielsweise eine sogenannte *Activator-Klasse* festgelegt werden, mit deren Hilfe auf die Aktionen des Plug-in Lebenszyklus reagiert werden kann. Dazu implementiert sie das Java-Interface *org.osgi.framework.BundleActivator* und überschreibt dessen Methoden *start* und *stop*, die aufgerufen werden, wenn das Plug-in aktiviert bzw. gestoppt wird.

Eine RCP-Anwendung, die in Form von Eclipse Projekten vorliegt, ist zunächst nicht ohne IDE lauffähig. Eine allgemein (ohne IDE) lauffähige Version dieser Anwendung erhält man durch einen sogenannten Buildprozess, in dem Bestandteile der Anwendung compiliert, zu JAR-Dateien verpackt und zu einer lauffähigen Gesamtanwendung zusammengestellt werden. (vgl. [Ebe11] S. 35) Für diesen Buildprozess muss ein sogenanntes *Product* für die Anwendung erstellt werden. Ein Product definiert die Bestandteile der zu buildenden RCP-Anwendung. Dies kann anhand von Features (Feature basierend) oder anhand von Plug-ins (Plug-in basierend) erfolgen. Darüber hinaus wird in einem Product festgelegt, wie diese Bestandteile zusammenzupacken und zu exportieren sind. (vgl. [Ebe11] S. 35) So besteht beispielsweise

die Möglichkeit eine Anwendung zu *branden*, d.h. mit Titel, Splash-Screen und Icons zu versehen. (vgl. [Ebe11] S. 35) Ein Product besteht aus zwei Teilen: einer Extension zu *org.eclipse.core.runtime.products* und einer .product-Datei. In der Extension sind Einstellungen zum Branding festgelegt. Die .product-Datei enthält hingegen eine Auflistung von den in der Anwendung verwendeten Features bzw. Plug-ins.

5.1.3 Das Eclipse Modeling Framework

Die grundlegenden Konzepte des Eclipse Modeling Framework (EMF) entspringen aus dem Ansatz der modellgetriebenen Softwareentwicklung (MDSD).

"Ein Modell beschreibt oder spezifiziert ein System zu einem bestimmten Zweck. Dazu erfasst es alle hierfür relevanten Aspekte, etwa Struktur, Verhalten, Funktion und Umwelt des Systems."([VG06]S. 24) In der herkömmlichen Softwareentwicklung werden Modelle meist als Werkzeug zur Beschreibung von Sachverhalten eingesetzt. In Abgrenzung dazu setzt man Modelle in der MDSD zur Definition von Softwarekomponenten ein, aus denen dann vollautomatisch verwendbare Softwarekomponenten generiert werden können. Hierbei müssen Modelle einige zusätzliche Eigenschaften erfüllen. Beispielsweise müssen sie in einer fest definierten Sprache beschrieben sein. Diese Sprache ist wiederum ein System und wird durch ein Modell definiert, welches Metamodell genannt wird.

Eine weitere wichtige Eigenschaft ist der formelle Charakter der erstellten und verwendeten Modelle. Ein formelles Modell ist ein Modell, das sich unter bloßer Kenntnis der Sprache mit der es beschrieben ist, deterministisch auswerten lässt. Dies ist eine notwendige Bedingung für die vollautomatische Auswertbarkeit von Modellen.

Die Umwandlung von einem Modell in ein anderes Modell wird *Modell-zu-Modell-Transformation*(M2M) genannt. (vgl. [WIK16f]) Eine Abbildung, die von der Menge der Komponenten des einen Modells auf die Menge von Komponenten des anderen Modells abbildet, wird Mappingregel genannt. Die Menge aller für die Transformation benötigten Mappingregeln heißt Mapping. Abbildung 36 verdeutlicht diesen Sachverhalt.

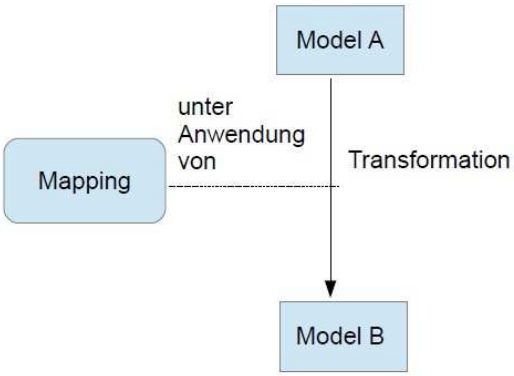


Abbildung 36: Darstellung einer M2M

Modelle lassen sich als plattformspezifisch oder als plattformunabhängig klassifizieren. Ein plattformunabhängiges Modell (PIM) beschreibt formal die Struktur und Funktionalitäten eines Systems, unabhängig

von der zugrundeliegenden Plattform. Ein plattformspezifisches Modell (PSM) hingegen ist mit plattformabhängigen Funktionen angereichert. Mittels Transformation kann ein PIM in ein PSM überführt werden. Dies ermöglicht es Strukturen und Funktionalitäten losgelöst von der konkreten Implementierung zu betrachten und in einem PIM festzuhalten. Dieses PIM kann dann in einer oder auch mehreren Plattformen verwendet werden, in dem man es in die entsprechenden plattformspezifischen Modelle überführt. Die folgende Abbildung 37 verdeutlicht dies.

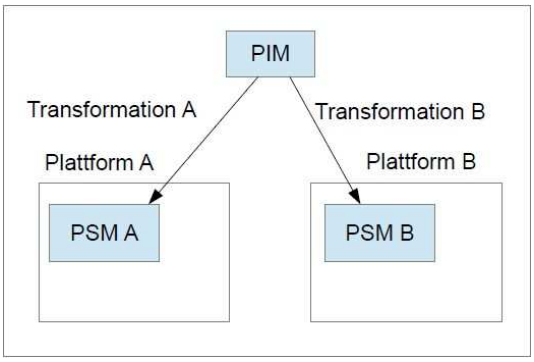


Abbildung 37: Darstellung Transformation PIM zu PSM

Ein PSM, das alle Informationen zu Konstruktion und Betrieb eines Systems definiert, ist eine Implementation. Man spricht in solchen Fällen auch von ausführbaren Modellen. (vgl. [VG06] S. 27) Modellgetriebene Softwareentwicklung ermöglicht es letztendlich, Software auf einen sehr hohen Abstraktionsniveau zu entwickeln, denn das Entwickeln von Softwarefragmenten erfolgt hierbei größtenteils durch das Erstellen von Modellen.

Das Eclipse Modeling Project ist ein Top-Level-Projekt der Eclipse Foundation (vgl.[EPR16]) zur Förderung und Zusammenführung von MDSD-Technologien innerhalb der Eclipse-Community. Außerdem "zeichnet sich EMF durch eine breite Unterstützung für die Interoperabilität mit anderen Werkzeugen aus. (vgl. [Hov10]) So können beispielsweise mittels des Graphical Modeling Frameworks graphische Editoren auf Basis von EMF und dem Graphical Editing Framework generiert werden. (vgl. [Hov10]) Das Eclipse Modeling Framework (EMF) ist Bestandteil des Eclipse Modeling Projects, mit dessen Hilfe sich die Datenmodelle plattformunabhängig modellieren und weiter verarbeiten lassen. EMF ist als Eclipse Plug-in konzipiert und lässt sich als solches nahtlos in die Eclipse IDE integrieren. (vgl. [VG06]S. 292)

Ein wichtiger Bestandteil von EMF ist das Ecore-Metamodell, das eine Sprache zur Beschreibung formaler plattformunabhängiger Modelle definiert. Modelle, die mit den Elementen dieser Sprache dargestellt sind, heißen Ecore-Modelle. Abbildung 38 gibt einen Überblick über die Kernelemente des Ecore-Metamodells.

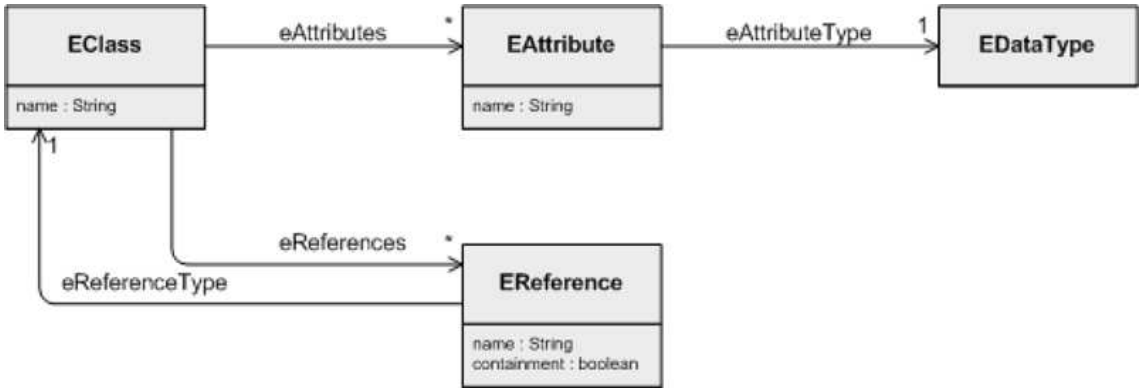


Abbildung 38: Kernelemente des Ecore-Metamodells (Quelle: [Poh10])

- EClass: Wird verwendet, um eine beliebige Klasse zu repräsentieren. Sie hat einen Namen sowie eine unbestimmte Anzahl von Attributen und Referenzen.
- EAttribute: Dient zur Repräsentation eines Attributs. Letzteres besteht aus einem Namen und einem Typ.
- EReference: Wird verwendet, um eine Assoziation zwischen Klassen zu beschreiben. Es enthält einen Namen, ein Flag, ob es sich um eine Einschlussbeziehung handelt, und einen Referenztyp, der auf eine Klasse zeigt.
- EDataType: Repräsentiert den Typ eines Attributs. (Beschreibungen übernommen aus [Poh10])

Ein Ecore-Modell kann demnach Strukturen von Daten in Form von Klassen und Beziehungen beschreiben. Ein hier nicht dargestelltes aber dennoch wichtiges Element des Ecore-Metamodells ist das EPackage. Es kann Elemente jedes Ecore-Typs enthalten. Jedes EPackage hat einen Namen und definiert durch ein Ns-Prefix-Attribut einen Namensraum für seine Elemente sowie einen Uniform Ressource Identifier (Abk. URI; engl. für „einheitlicher Bezeichner für Ressourcen“) für diesen Namensraum durch ein Ns-URI-Attribut. Ein EPackage kann beliebig viele EPackages enthalten, aber jedes Ecore-Modell hat genau ein EPackage, in dem all seine Elemente enthalten sind. Es ermöglicht somit die Trennung von Namensräumen innerhalb eines Ecore-Modells. Abbildung 39 gibt hierfür ein einfaches Beispiel.

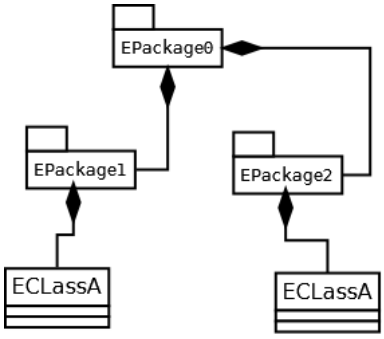


Abbildung 39: Beispiel Ecore-Modell

Das hier dargestellte Ecore-Modell besteht aus 3 EPackages und 2 EClasses. Das übergeordnete EPackage ist *EPackage0*, es enthält die beiden EPackages *EPackage1* und EPackage2. *EPackage1* und *EPackage2* besitzen jeweils eine ECLass mit den Namen *ECLassA*. Die Existenz zweier EKlassen mit den gleichen Namen ist durch die mit den *EPackage1* und *EPackage2* definierten Namensräume möglich.

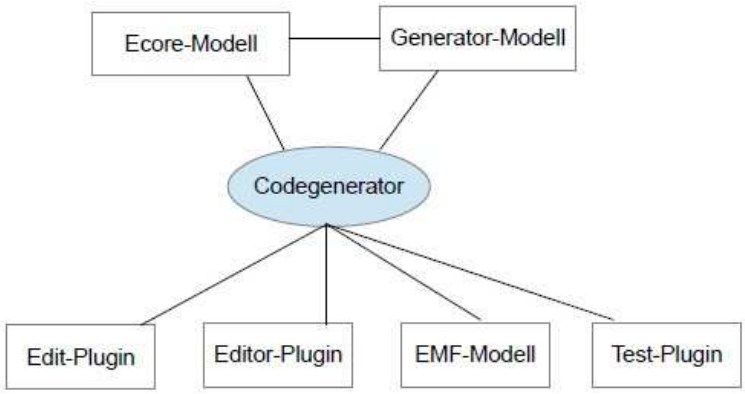


Abbildung 40: Diagramm der wichtigsten Elemente der EMF-Codegenerierung

Abbildung 40 gibt einen Überblick über die Weiterverarbeitung von Ecore-Modellen. Ein Ecore-Modell kann als Grundlage eines sogenannten Generator-Modells dienen. Das Generator-Modell erweitert ein Ecore-Modell um Informationen der Codegenerierung, beispielsweise in welchen Verzeichnis der zu generierende Code abgelegt werden soll. Mit einem ebenfalls in EMF enthaltenen Codegenerator kann das Generator-Modell dann wahlweise zu einem sogenannten EMF-Modell, einem Edit-Plug-in, einem EMF-Editor-Plug-in, oder zu einem Test-Plug-in transformiert werden. (vgl. [Vol16]) Hierfür stellt EMF eine Reihe von Wizards zur Verfügung. Ein EMF-Modell ist eine vollständige Java Implementierung eines Ecore-Modells. Es besteht beispielsweise aus

- Schnittstellenklassen für alle modellierten Objekte
- Implementierungsklassen für alle Schnittstellen
- Fabrikklassen zur Erzeugung von Exemplaren der Modellobjekte
- Paketklassen, die Zugriff auf die Metadaten des Modells gewähren (vgl. [TK]S. 9)

Das Edit-Plug-in stellt eine Reihe von Editierfunktionalitäten für einen baumbasierten Editor wie Undo/Redo für das EMF-Model bereit. Das EMF-Editor-Plug-in enthält einen Editor, mit dem man Instanzen des EMF-Modells erstellen und bearbeiten kann. (vgl. [GT05]S. 2). Der Editor greift hierbei auf die Funktionalitäten des Edit-Plug-ins zurück.

EMF ermöglicht es, Instanzen von EMF-Modellen in einer XML Datei zu speichern. Dies kann über ein Objekt der Klasse *XMLResource* geschehen, indem ihm die Objekte der EMF-Modell-Instanz und ein Dateipfad übergeben werden und im Anschluss daran seine XMLResource.save-Methode aufgerufen wird.

5.2 Bedienung und Benutzeroberfläche

Nach der Erläuterung der Technologien, mit dem der Prototyp implementiert wurde, soll nun der Prototyp aus der Sicht des Benutzers vorgestellt werden. Abbildung 41 ist ein Screenshot des Hauptfensters der Prototyp-Anwendung. Einige Bereiche und Bedienelemente wurden rot markiert und nummeriert.

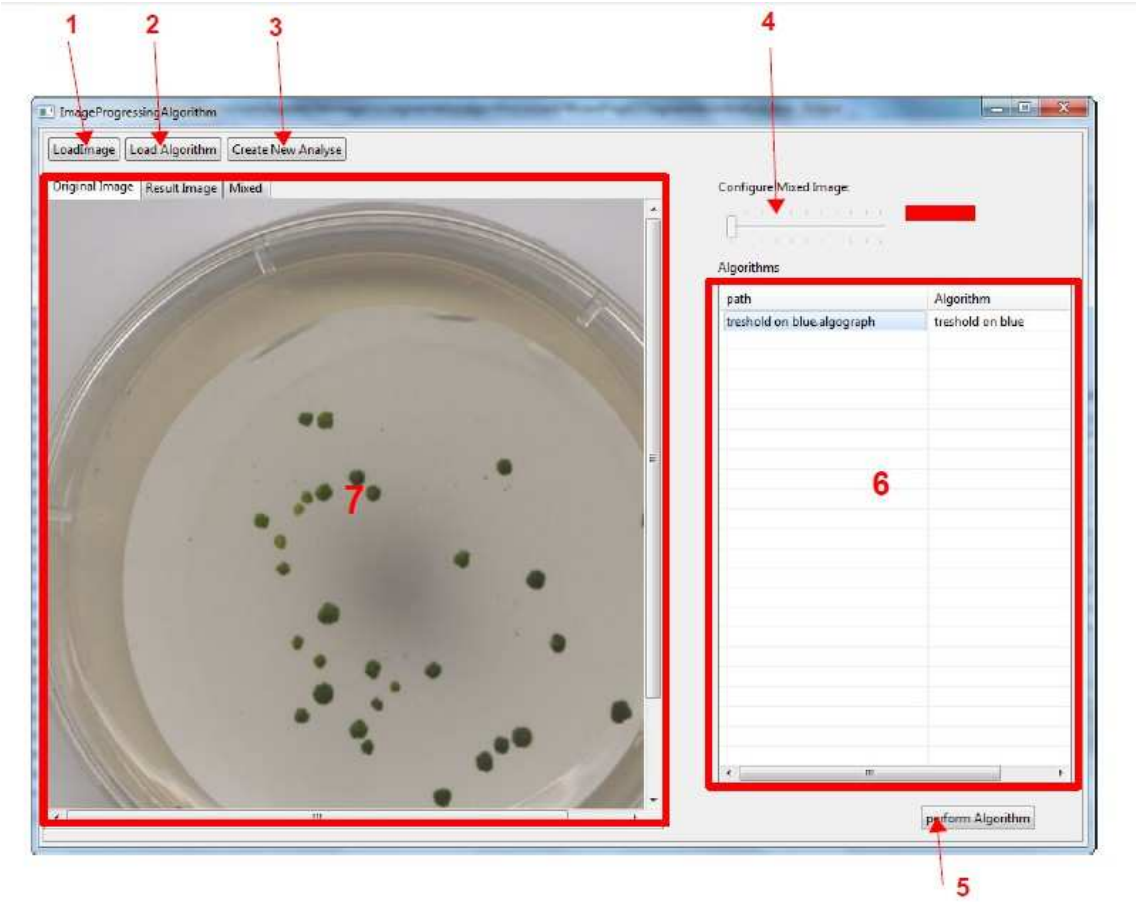


Abbildung 41: Screenshot Hauptfenster

Nach dem Programmstart öffnet sich das Hauptfenster. Nun kann der Benutzer mit einem Maus-Klick auf den *SWT-Button* "LoadImage" (1) ein Eingabebild aus dem Dateisystem auswählen. Das geschieht über einen sogenannten *SWT-FileDalog*, der sich nach Betätigen des "LoadImage"-Buttons öffnet und in dem der Benutzer durch das Dateisystem navigieren kann. Abbildung 42 zeigt den FileDialog zur Auswahl eines Eingabebildes. Hat der Benutzer ein Bild ausgewählt, so wird es im Tab "Original Image" des Tabfolders (7) angezeigt. Nun kann mit Klick auf den Button "Load Algorithm" (2) ein konfigurierter Algorithmengraph im Dateisystem ausgewählt werden. Dies erfolgt wie bei der Wahl eines Eingabebildes über einen SWT-FileDialog. Der geladene Dateipfad und der Name des Algorithmengraphen werden in der Tabelle (6) aufgelistet. Mit Klick auf den Button "perform Algorithm" (5) wird der Algorithmengraph auf dem Eingabebild ausgeführt.

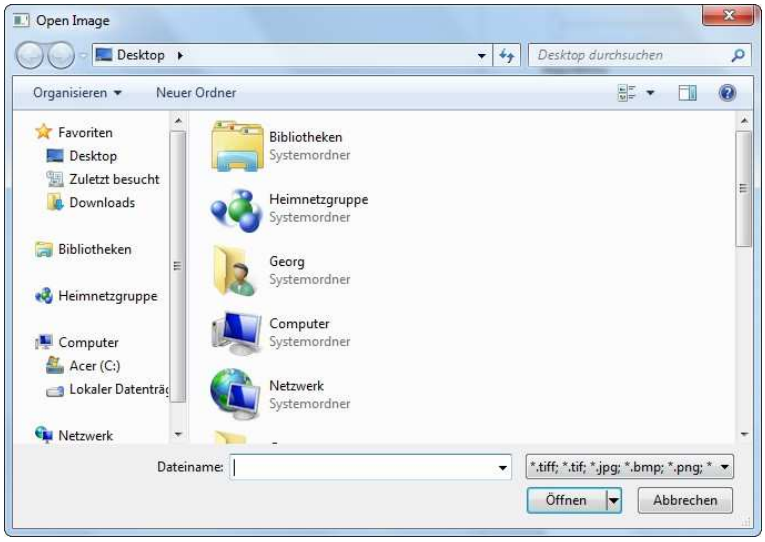


Abbildung 42: FileDialog zur Auswahl eines Eingagebildes

Im Tab "Result Image" des Tabfolders (7) wird das Ergebnis der Ausführung des Algorithmengraphen als Binärbild angezeigt. Im Tab "Mixed" kann das Eingabebild mit den ermittelten Objektflächen, den schwarzen Pixel des Ergebnisbildes, überblendet werden. Dabei kann der Grad der Transparenz der Objektflächen über den Schieberegler (4) geregelt werden.

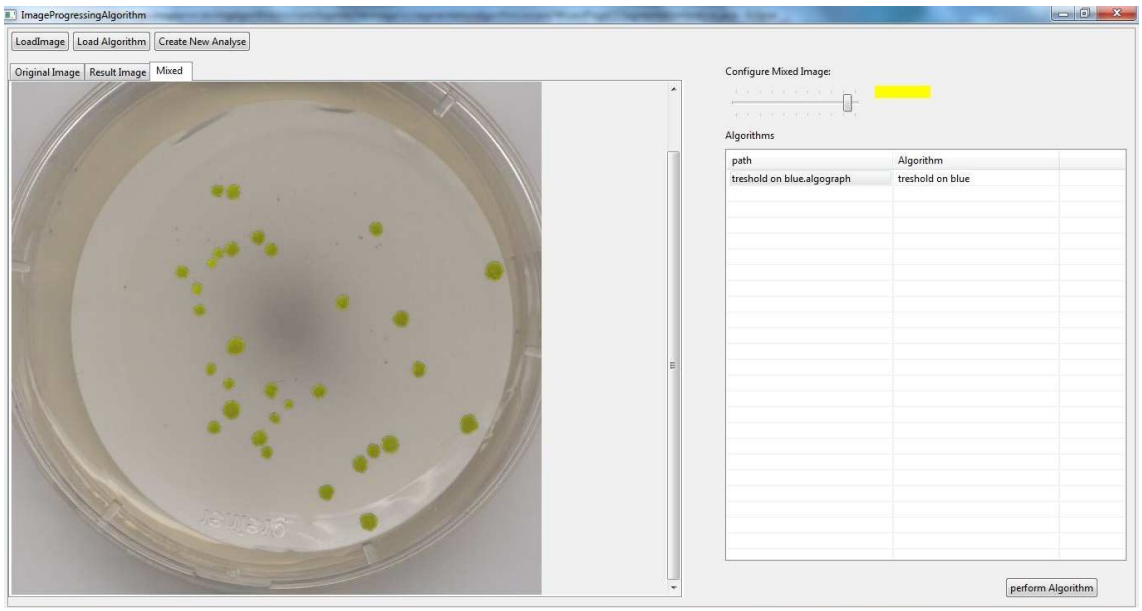


Abbildung 43: Eingabebild mit überblendetem Ergebnisbild

Abbildung 43 zeigt das Hauptfenster mit ausgewähltem "Mixed"-Tab, in dem das Originalbild mit den Objektflächen überlagert wird. Die Objektflächen sind dabei gelb markiert. Die Farbe der Objektflächen kann mit Klick auf das farbige Rechteck neben dem Schieberegler zur Einstellung der Transparenz geändert werden.

Die Analyse von Algorithmengraphen kann mit Klick auf den Button “Create New Analyse” (3) initiiert werden. Dies geschieht über ein Wizard, der mit Betätigen des Buttons aufgerufen wird. Die Analyse von Algorithmengraphen erfolgt in drei Schritten:

- Erstellen des a priori Wissens
- Durchführung der Analysen für alle Algorithmengraphen für alle Klassen
- Auswerten der Analysen und Persistieren der konfigurierten Algorithmengraphen

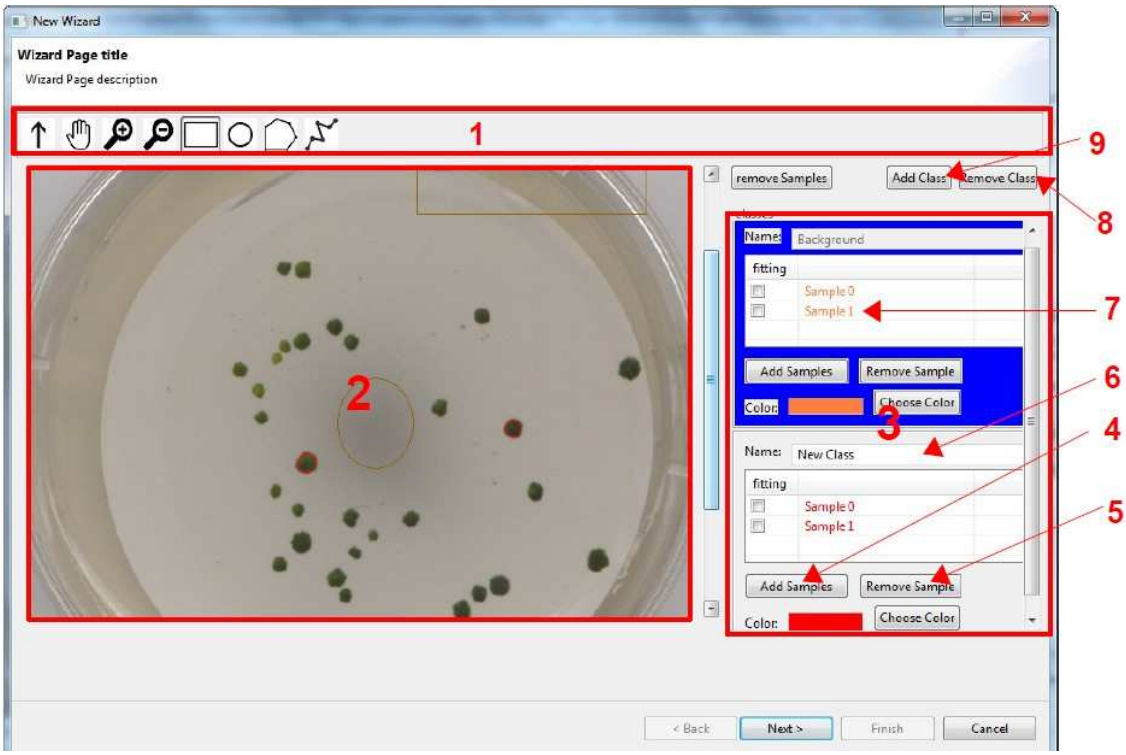


Abbildung 44: Wizard-Seite zur Konfiguration des a priori Wissens

Abbildung 44 zeigt die Wizard-Seite zur Definition des a priori Wissens. In der Canvas (2) wird das Originalbild angezeigt. Auf ihm können mittels der Werkzeuge der Toolbar (1) verschiedene geometrische Objekte erstellt werden. Diese können dann durch Klick auf den Button “Add Samples” (5) einer Segmentierungsklasse (Klasse) zugeordnet werden. Die Klassen werden in den mit einem roten Rechteck umrandeten Bereich 3 angezeigt. Die Samples jeder Klasse werden in der Tabelle der jeweiligen Klasse aufgelistet. Mit dem “Remove Sample”-Button einer Klasse können einzelne Samples aus einer Klasse entfernt werden. Über das Textfeld “Name” (4) kann der Bezeichner einer Klasse festgelegt werden. Mit den Button “Add Class” (9) kann eine neue Klasse angelegt werden und mit dem Button “remove Class” (8) kann eine Klasse entfernt werden. Wurde das a priori Wissens hinreichend definiert, so kann mit dem Klick auf den “Next”-Button des Wizards die Wizard-Seite zur Ausführung der Analyse aufgerufen werden.

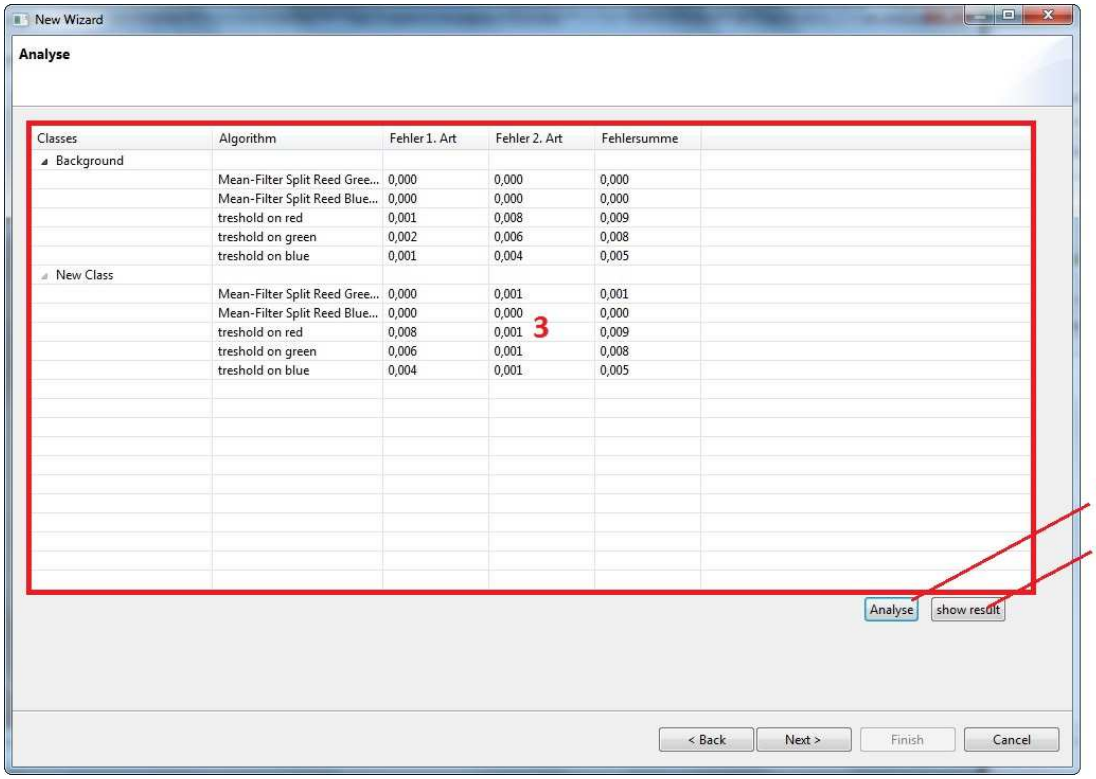


Abbildung 45: Wizard-Seite zur Ausführung der Analyse

Abbildung 45 zeigt die Wizard-Seite zur Ausführung der Analyse. In der Tabelle (3) werden für jede Klasse alle implementierten Algorithmengraphen aufgelistet. Die Zellen der Spalte “Algorithms” enthalten die Namen der Algorithmengraphen. In den Spalten “Fehler 1. Art”, “Fehler 2. Art” und “Fehlersumme” wird das Ergebnis der Segmentierungsfehlerberechnung der Analyse angezeigt. Mit Klick auf den Button “Analyse” (1) wird die Analyse durchgeführt. Mit dem Button “show result” (2) kann das Ergebnisbild des mit der Analyse konfigurierten Algorithmengraphen, der in der Tabelle ausgewählt wurde, angezeigt werden. Dabei öffnet sich ein neues Fenster, wie es in Abbildung 46 zu sehen ist.

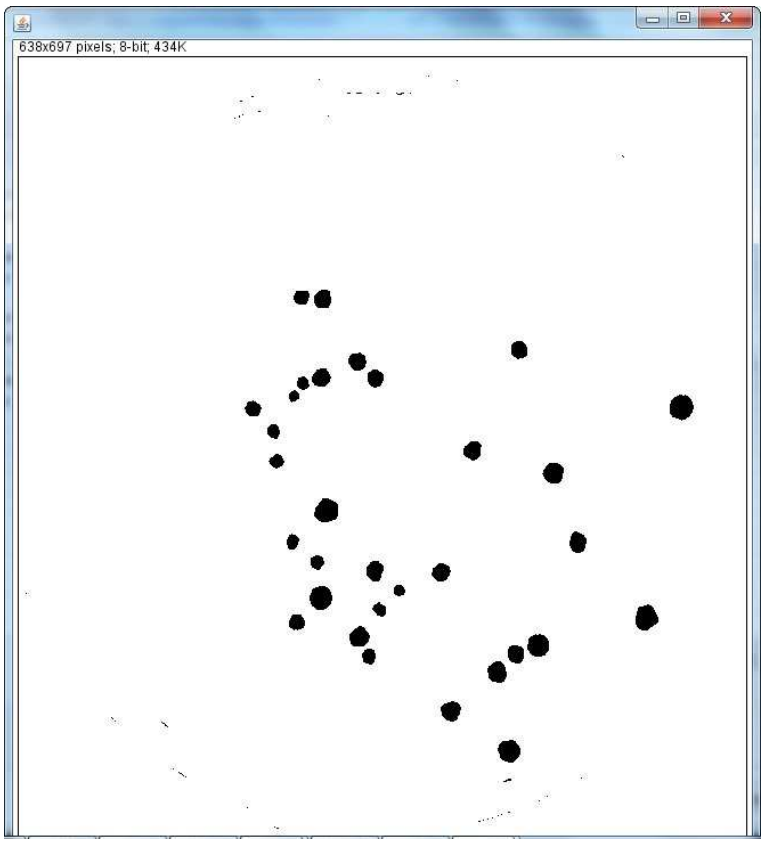


Abbildung 46: Ergebnisbild einer Analyse

Nach Durchführung der Analyse kann mit Klick den “Next”-Button des Wizards die Wizard-Seite zum Auswerten der Analysen und Persistieren der konfigurierten Algorithmengraphen aufgerufen werden. Abbildung 47 ist ein Screenshot dieser Wizard-Seite. Im Bereich 1 wird das Eingabebild angezeigt. In der Tabelle 3 werden die Klassen und in Tabelle 4 die Analyse-Ergebnisse einer in der Tabelle 3 ausgewählten Klasse aufgelistet. Mit der Wahl eines der Analyse-Ergebnisse in Tabelle 4 wird das Eingabebild mit dem Ergebnis der Ausführung des jeweiligen mit der Analyse konfigurierten Algorithmengraphen überblendet. Dafür kann der Grad der Transparenz der Ergebnis-Regionen mit dem Schieberegler 2 festgelegt werden.

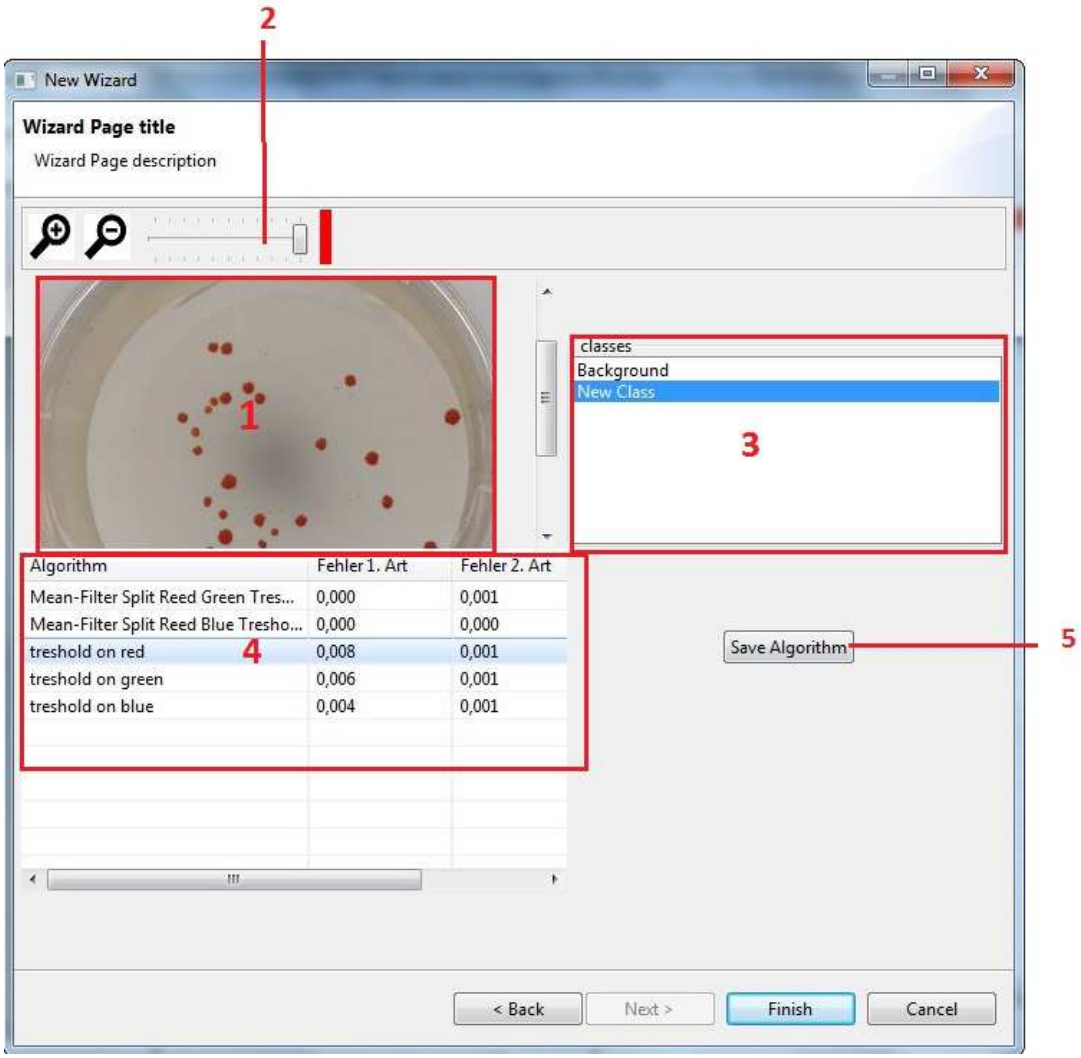


Abbildung 47: Wizard-Seite zur Auswertung der Analyse-Ergebnisse

Mit Klick auf den Button “Save Algorithm” (5) kann ein in Tabelle 4 ausgewählter konfigurierter Algorithmengraph persistiert werden. Dafür wird ein FileDialog aufgerufen, mit dem ein Verzeichnis und ein Dateiname festgelegt werden kann. Mit Betätigen des “Finish”-Buttons des Wizard-Dialogs wird der Wizard-Dialog geschlossen und das Hauptfenster wird angezeigt.

5.3 Implementierung der Modelle des Algorithmengraphen und des Analysegraphen

Die Implementierung der Modelle des Algorithmengraphen und des Analysegraphen wurde mit dem im Abschnitt 5.1.3 vorgestellten EMF-Framework umgesetzt. Dies ist zum einen damit begründet, dass so auf einem hohen Abstraktionsniveau vorgegangen werden konnte und zum anderen die Persistenzfunktionalitäten des EMF-Framworks für das Speichern von konfigurierten Algorithmengraphen verwendet werden können.

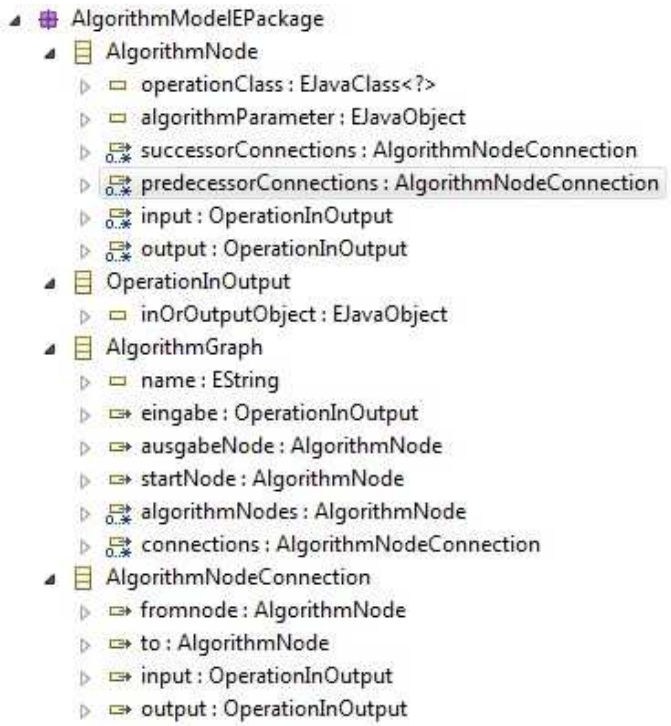


Abbildung 48: Algorithmengraph im Ecore-Editor

Abbildung 48 zeigt einen Auschnitt des EMF-Editors mit den Ecore-Modell des Algorithmengraphen und bietet somit eine Übersicht über dessen Klassenstruktur. Es wurde sich bei der Implementierung weitestgehend an das im Abschnitt 4.2 vorgestellte Modell gehalten. Die Bezeichner der Klassen und Attribute liegen allerdings gemäß der Namenskonventionen Kapelans in englischer Sprache vor. So werden Operationen in dieser Implentierung beispielsweise *AlgorithmNode* genannt. Ein Unterschied in der Struktur des Ecore-Modells ist, dass die Klasse *AlgorithmNodeConnection* Attribute to und from für die Referenzen der Nachfolger-Knoten und Vorgänger-Knoten hat. Dies ermöglicht ein leichteres Zuordnen der Ein- und Ausgaben (input, output), die mit der jeweiligen Verbindung verbunden werden zu den zugehörigen *AlgorithmenNode-Objekten*. Im Algorithmengraph-Modell des Abschnitts 4.2 wurden die konkreten Ausprägungen der Operationen mit dem Instrument der Vererbung modelliert. In dieser Implementierung wird die algorithmische Funktionalität einer Operation mittels einer Java-Klasse umgesetzt, die dem Attribut operationClass eines AlgorithmNode-Objekts zugewiesen werden kann. Auf diese Weise muss für die Implementierung einer neuen Operation keine Änderung des Ecore-Modells erfolgen, sondern lediglich eine

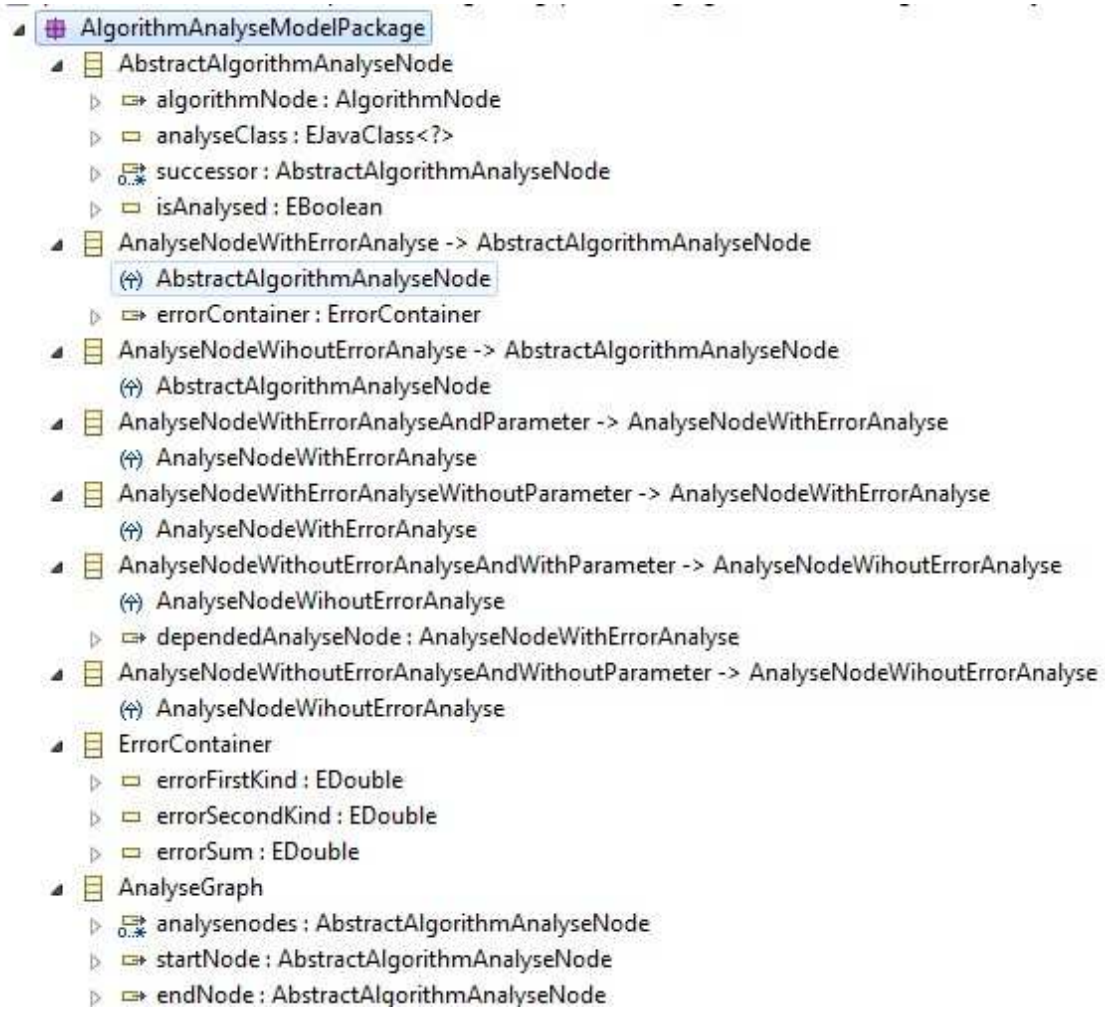


Abbildung 49: Analysegraph im Ecore-Editor

neue Operationen-Klasse in Java implementiert werden.

Abbildung 49 zeigt einen Ausschnitt des EMF-Editors mit dem Ecore-Modell des Analysegraphen. Auch hier wurden englische Bezeichner verwendet. Im Vergleich zu dem im Abschnitt 4.3 vorgestellten Modell des Algorithmengraphen, findet man in dieser Implementierung keine Platzhalter für eine konkrete AnalyseOperation oder einen ParameterGenerator. Das Zuordnen von AnalyseOperation-Objekten und ParameterGenerator-Objekten zu *AnalyseNode*-Objekten erfolgt in dieser Implementierung anhand der mit dem operationClass-Attribut zugeordneten Operation des dem jeweiligen AnalyseNode-Objekt zugewiesenen AlgorithmNode-Objekts. Dies vereinfacht das Modellieren von Analysegraphen, da so die Operations-spezifischen Eigenschaften des Algorithmengraphen und des Analysegraphen nur an einer Stelle festgelegt werden müssen.

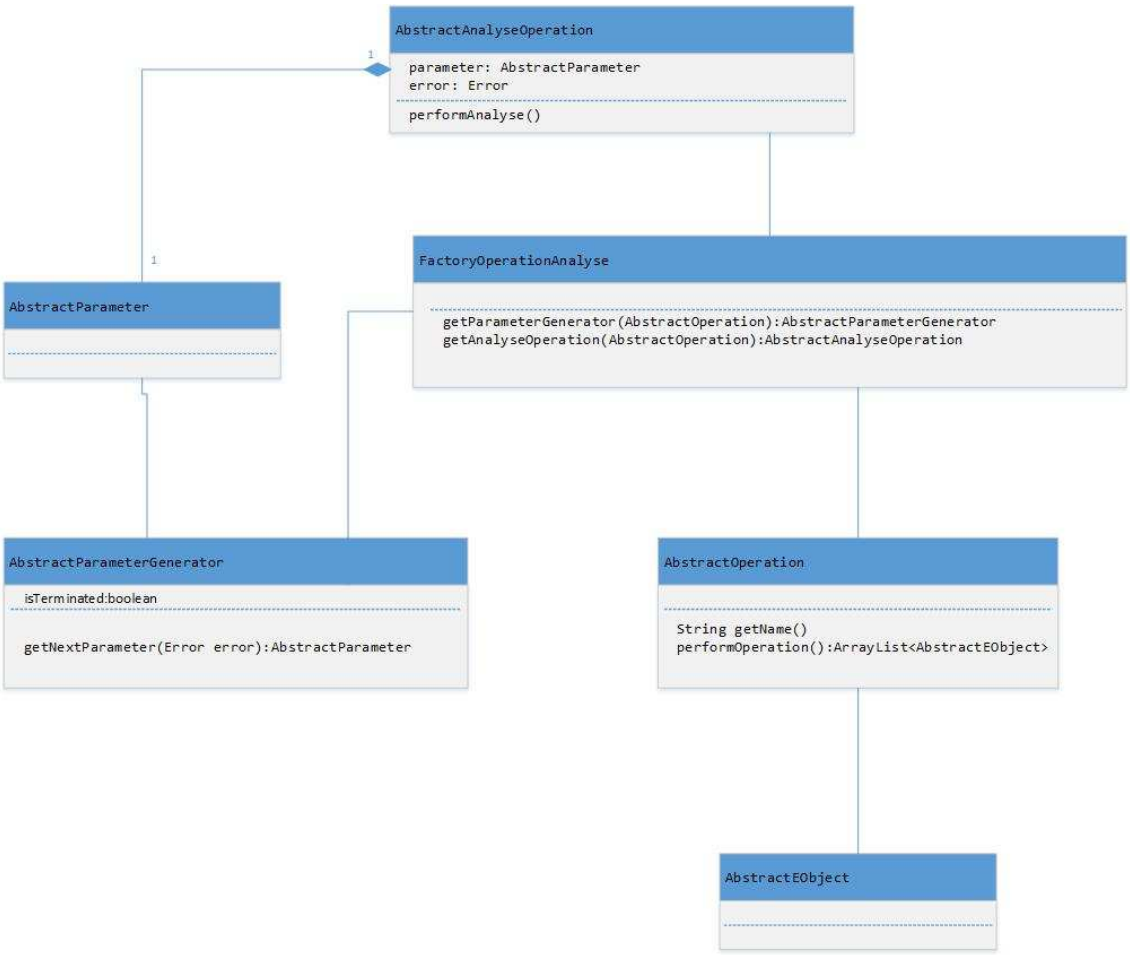


Abbildung 50: Klassendiagramm der Operationen

Abbildung 50 gibt einen Überblick über die Java-Klassenstruktur der Operationen-, der AnalyseOperation- und der ParameterGenerator-Klassen der Implementierung. Neben der im Vergleich zu den im Abschnitt 4 vorgestellten Modellen veränderten Namensgebung dieser Klassen , fällt auf, dass die Klassen mit einer *FactoryOperationAnalyse*-Klasse assoziiert sind. Mit der FactoryOperationAnalyse-Klasse lassen sich in Abhängigkeit von AbstractOperation-Objekten die Objekte der Operations-spezifischen ParameterGenerator- und AnalyseOperationen-Klassen erzeugen. Dies kann über das Aufrufen der Methoden *getParameterGenerator* und *getAnalyseOperation* der FactoryOperationAnalyse-Klasse geschehen. Die Implementierung einer neuen Operation kann wiefolgt erfolgen:

- Erstellen einer Klasse, die von AbstractOperation erbt und deren *getName*- und *performOperathion*-Methode überschreibt.
- Wenn nötig: Erstellen einer Operations-spezifischen Parameter-Klasse, die von AbstractParameter abgeleitet ist.
- Wenn nötig: Erstellen einer Operations-spezifischen ParameterGenerator-Klasse, die von AbstractParameterGenerator erbt und deren *getNextParameter*-Methode überschreibt.

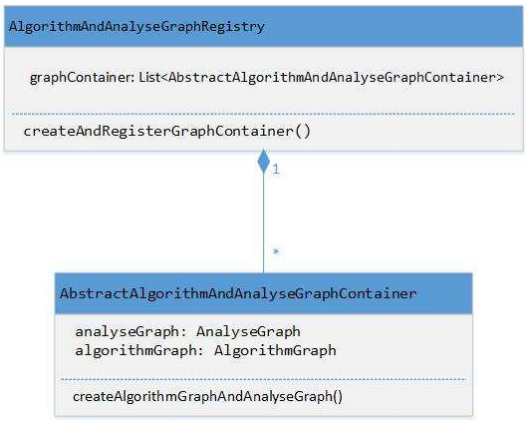


Abbildung 51: Klassendiagramm AlgorithmAndAnalyseGraphRegistry und AbstractAlgorithmAndAnalyseGraphContainer

- Wenn nötig: Erstellen einer Operations-spezifischen AnalyseOperation-Klasse, die von AbstractAnalyseOperation abgeleitet ist und deren performAnalyse-Methode überschreibt.
- Wenn nötig: Modifizierung der getParameterGenerator- oder der getAnalyseOperation-Methode der FactoryOperationAnalyse Klasse, so dass sie die Operations-spezifischen ParamenterGenerator- oder AnalyseOperator-Objekte zurück geben.

5.4 Implementierung von Algorithemgraphen und Analysegraphen

Algorithmengraphen und Analysegraphen werden zur Laufzeit erzeugt. Abbildung 51 gibt einen Überblick über die Klassenstruktur der dabei beteiligten Java-Klassen.

Die Klasse AbstractAlgorithmAndAnalyseGraphContainer repräsentiert eine abstrakte Strukur zur Definition konkreter Algorithmen- und den zugehörigen Analysegraphen. Zur Implementierung eines Algorithmengraphen und dem ihm zugeordneten Analysegraphen, wird eine von der AbstractAlgorithmAndAnalyseGraphContainer-Klasse abgeleite Klasse erstellt und *createAlgorithmGraphAndAnalyseGraph* methode überschieben. Alle implementierten Algorithmen- und Analysegraphen werden zentral in einer Klasse *AlgorithmAndAnalyseGraphRegistry* regestriert. Neue Algorithmen- und Analysegraphen können ihr mit Modifizierung ihrer *createAlgorithmGraphAndAnalyseGraph*-Methode bekannt gemacht werden.

5.5 Implementierung der Ausführungsalgorithmen

Die im Abschnitt 4.4 vorgestellten Ausführungsalgorithmen für Algorithmen- und Analysegraphen sind in den beiden Java-Klassen *InterpreterAlgorithmGraph* und *InterpreterAnalyseGraph* implementiert.

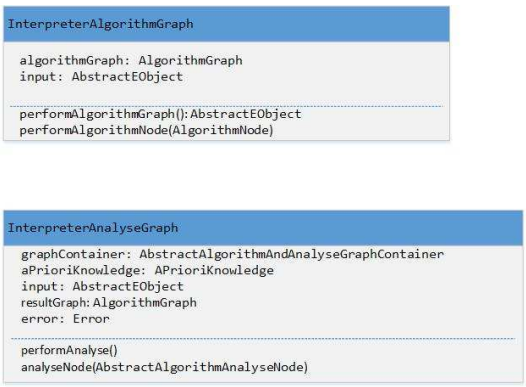


Abbildung 52: Klassendiagramm InterpreterAlgorithmGraph und InterpreterAnalyseGraph

Abbildung 52 gibt einen Überblick über diese beiden Klassen. Zur Ausführung eines konfigurierten Algorithmengraphen ist ein InterpreterAlgorithmenGraph-Objekt zu instanziieren, dem *algorithmGraph*-Attribut ein AlgrithmGraph-Objekt und dem *input*-Attribut eine Instanz des AbstractEObject mit dem Eingabebild zuzuweisen und anschließend die *peformAlgorithmGraph*-Methode aufzurufen. Die peformAlgorithmGraph-Methode gibt ein AbstractEObjekt-Objekt mit dem Ergebnisbild zurück. Zur Ausführung einer Analyse wird ein InterpreterAnalyseGraph-Objekt instanziiert. Seinem graphContainerAttribut wird ein AbstractAlgorithmGraphContainer-Objekt, seinem aPrioriKnowledge-Attribut ein ApriKnowledge-Objekt und seinem input-Attrubut ein AbstractEObject mit dem Eingabebild zugewiesen. Danach wird die performeAnalyse-Methode aufgerufen und die Analyse wird ausgeführt. Dabei werden den Attributen *error* und *resultGraph* die Ergebnisse der Analyse zugeordnet.

5.6 Performancetests

Nachdem in den vorangegangen Ausführungen des Abschnitts 5 die Umsetzbarkeit des SM-Verfahrens anhand einer beispielhaften Implementierung verdeutlicht wurden, soll in diesem Abschnitt die Eignung des SM-Verfahrens für den praktischen Einsatz anhand einiger durchgeführter Perforfmancetests untersucht werden. Hierfür wurden der im Abschnitt 4 als Beispiel vorgestellte Algorithmengraph und dessen zugehöriger Analysegraph implementiert. Diese sind in Abbildung 53 noch einmal schematisch dargestellt.

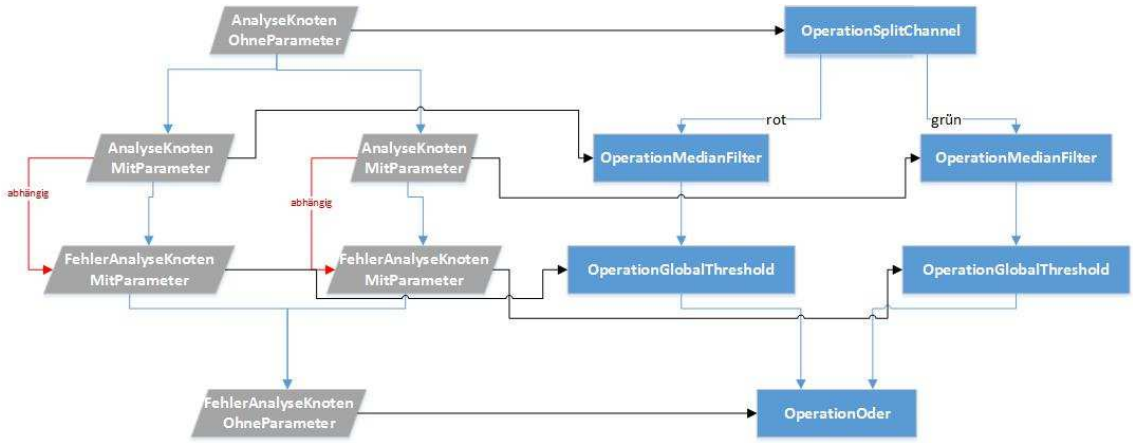


Abbildung 53: Schematische Darstellung von einem Algorithmengraph und dem zugehörigen Analysegraph

Vier Bilder unterschiedlicher Größe wurden mit dem Analysegraph analysiert. Im Anschluss daran wurde der mit der Analyse konfigurierte Algorithmengraph mit jedem Bild ausgeführt. Tabelle 2 gibt einen Überblick über die Größe der Bilder und die Dauer der Analysen und Ausführungen.

	Bild 1	Bild 2	Bild 3	Bild 4
Breite in px	312	638	1137	5312
Höhe in px	227	697	928	2998
Anzahl Pixel (AZ)	70824	444686	1055136	12206976
Dauer Analyse (TN)	290 ms	1582 ms	3371 ms	76,38 s
Dauer Ausführung (TS)	30 ms	120 ms	290 ms	4,6 s
AZ/TN	244	281	316	159
AZ/TS	2360	3705	3638	2653

Tabelle 2: Testergebnisse Performancetest

Allgemein lässt sich feststellen, dass die Dauer des Analyse-Prozesses sehr viel größer ist als die des Ausführungs-Prozesses. Mit der Größe der Bilder nimmt auch die Analyse- und Ausführungsdauer zu. Zwischen der Anzahl der Pixel und der Analysedauer scheint ein linearer Zusammenhang zu bestehen. Gleiches gilt für die Ausführungsdauer. Während die Ausführung bei dem größten Bild 4,6 Sekunden dauert und damit durchaus akzeptabel ist, scheint die Analysedauer mit 76,38 Sekunden unangemessen groß, wenn man bedenkt, dass auch mehrere Analysegraphen untersucht werden sollen. So würde die Analyse von beispielsweise zehn Analysegraphen, unter der Annahme, dass der Rechenaufwand bei jedem Analysegraphen in etwa gleich groß ist, eine Gesamtdauer von über zehn Minuten betragen. Bei dem nächst kleineren Bild (Bild 3) würde die Gesammtrechendauer für die Durchführung von zehn Analysen mit ähnlichen Aufwand hingegen in etwa 33 Sekunden betragen und wäre damit für den Endanwender akzeptabel.

Es ist bei der Betrachtung dieser Testreihe zu berücksichtigen, dass der untersuchte Analysegraph keineswegs repräsentativ sein muss. Die Analyse- und Ausführungsdauer hängt neben der Bildgröße auch von den jeweiligen Operationen und deren Kombination ab.

6 Auswertung und Ausblick

Wie in dieser Arbeit verdeutlicht wurde, gibt es eine Vielzahl verschiedener Ansätze und Verfahren zur Segmentierung von Bildern. Das SM-Verfahren bietet eine Möglichkeit diese miteinander zu vereinen. Für den Endbenutzer bedeutet das eine einfache und standartisierte Herangehensweise zur Segmentierung. So besteht die Konfiguration des SM-Verfahrens lediglich darin Klassen von Regionen anzulegen und ihnen Samples zuzuordnen. Es bedarf darüber hinaus keinerlei Kenntnis über die Wirkweise, Eignung und Konfigurationsmöglichkeiten von Bildverarbeitungsalgorithmen.

Fachwissen auf dem Gebiet der Bildverarbeitung wird lediglich zur Implementierung neuer Algorithmen- und Analysegraphen sowie neuer Operationen benötigt. Das Erstellen neuer Algorithmen- und Analysegraphen mit bereits vorhandenen Operationen ist dabei jedoch wenig aufwendig. Denn hierzu bedarf es lediglich einer sinnvollen Kombination dieser Operationen. Für die Umsetzung von Bildsegmentierungsverfahren in Operationen für das SM-Verfahren, müssen gegebenenfalls auch operationspezifische Analyse-Operationen angelegt werden. Herfür bedarf es der Entwicklung geeigneter Strategien zur Untersuchung des Parameter-Raums der jeweiligen Operation. Dies könnte ein Ansatzpunkt für weitergehende Forschungen sein.

Das Erstellen und Ausführen von Analysegraphen ist eine Möglichkeit zur Untersuchung des Parameterraums von Bildsegmentierungsalgorithmen. Dies könnte unter Beibehaltung des Segmentierungsfehlers als Bewertungskriterium durch andere Verfahren ersetzt werden. So ist beispielsweise zu prüfen, ob sich nicht auch ein evolutionärer Ansatz für die Lösung dieser Problemstellung eignen könnte.

Anhand einer prototypischen Implementierung wurde die Umsetzbarkeit des SM-Verfahrens demonstriert. Diese diente auch als Grundlage für die Durchführung von Performancetests. Dabei zeigte sich, dass die Ausführung konfigurierter Algorithmengraphen in hinreichend großer Geschwindigkeit gewährleistet ist. Die Analyse ist bei sehr großen Bildern jedoch sehr aufwendig, so dass sie unverhältnismäßig viel Zeit in Anspruch nimmt. Dem könnte man Abhilfe schaffen, in dem man große Bilder mittels Skalierung zu kleineren Bildern transformiert. Oft ist dies möglich ohne dass für die Segmentierung relevante Informationen verloren gehen. Ist das nicht der Fall, dann könnte man für die Analyse beispielsweise nur ausgewählte Bereiche des Eingabebildes verwenden.

Bei der zukünftigen Integration des SM-Verfahrens in die Software-Produkte von Kapelan wird sich zeigen, in wie weit es in der Praxis sinnvoll eingesetzt werden kann.

Index

Literatur

[BRO] *Textursammlung von Brodatz*. http://www.ux.uis.no/~tranden/brodatz.html

[CCE16] *Heterotrophic Plate Count*. http://www.moldbacteriaconsulting.com/bacteria/heterotrophic-plate-count-what-is-hpc-and-when-is-the-right-time-to-use-it.html. Version: 03 2016

[CD98] Christan Demant, Peter W. Bernd Streicher-Abel: *Industrielle Bildverarbeitung: Wie optische Qualitätskontrolle wirklich funktioniert*. Springer-Verlag, 1998

[Ebe11] Ebert, Ralf: *Eclipse RCP Entwicklung von Desktopt Anwendungen mit der Eclipse Rich Client Plattform 3.7*. 2011

[EDO16] *Eclipse Download Seite*. http://www.eclipse.org/downloads/. Version: 03 2016

[EPR16] *Eclipse: Projecte*. http://www.eclipse.org/projects/listofprojects.php. Version: 03 2016

[Erh08] Erhardt, Angelika: *Einführung in die Digitale Bildverarbeitung: Grundlagen, Systeme und Anwendungen*. Vieweg+Teubner | GWV Fachverlag GmbH, 2008

[FIL] http://lmb.informatik.uni-freiburg.de/people/haasdonk/DBV_FHO/DBV_FHO_SS08_E08.pdf

[GT05] Gabriele Taenzer, Karsten E. Claudia Ermel E. Claudia Ermel: *Erstellung eines graphischen Editor-Plug-Ins mit Eclipse EMF und GEF*. http://www.sigs.de/publications/os/2005/02/ehrig_ermel_OS_02_05.pdf. Version: 2005

[Hov10] Hovath, Zoltan: *Übersicht Eclipse Modeling Project EMP*. www.oio.de/public/opensource/uebersicht-eclipse-modeling-framework-emf-artikel.htm. Version: 2 2010

[KAP16a] *Kapelan: CC*. http://www.kapelanbio.de/produkte/labimage/labimage-cc.html. Version: 03 2016

[KAP16b] *Kapelan: Historie*. http://www.kapelanbio.de/unternehmen/historie.html. Version: 03 2016

[KAP16c] *Kapelan: LabImager TR*. http://www.kapelanbio.de/produkte/reader.html. Version: 03 2016

[Poh10] Pohl, Thomas: *Modellgetriebene Softwareentwicklung mit EMF und Xtext*. www.heise.de/developer/artikel/Modellgetriebene-Softwareentwicklung-mit-EMF-und-Xtext-912436.html. Version: 1 2010

[RG02] Rafael Gonzalez, Richard E. W.: *Digital Image Processing*. Prentice Hall Press, 2002

[SCH16] *Schulinformatik: Hough Transformation*. http://schulinformatik.info/?page_id=159. Version: 04 2016

[SEB15] *Skript zur Vorlesung Digitale Bildverarbeitung des Instituts für Neuroinformatik an der Ruhr-Universität Bochum*. http://www.ini.rub.de/uploads/document/attachment/305/KantenKonturenHough.pdf. Version: Sebastian Houben 2015

[TK] Timo Kehrer, Cristoph B.: *Einführung in das Eclipse Modeling Framework*. http://pi.informatik.uni-siegen.de/Mitarbeiter/kehrer/lehre/ws10/st1/emf-intro/emf.pdf

[TL97] Thomas Lehmann, Erich Pelikan Rudolf R. Walter Oberschelp O. Walter Oberschelp: *Bildverarbeitung für die Medizin*. Springer-Verlag, 1997

[VG06] Volker Gruhn, Carsten R. Daniel Pieper P. Daniel Pieper ; 1 (Hrsg.): *MDA*. Springer-Verlag Heidelberg, 2006

[Vol16] Volgel, Lars: *Eclipse Modeling Framework (EMF) - Tutorial*. http://www.vogella.com/articles/EclipseEMF/article.html. Version: 04 2016

[WB16] Wayne Beaton, Jonas Helming und Maximilian K.: *Was ist Eclipse? Teil 2*. http://it-republik.de/jaxenter/artikel/Was-ist-Eclipse-Teil-2-4132.html. Version: 03 2016

[WIK16a] *Wikipedia: Dreifarbentheorie*. https://de.wikipedia.org/wiki/Dreifarbentheorie. Version: 03 2016

[WIK16b] *Wikipedia: Farbkanal*. https://de.wikipedia.org/wiki/Farbkanal. Version: 04 2016

[WIK16c] *Wikipedia: Histogramm*. https://de.wikipedia.org/wiki/Histogramm. Version: 04 2016

[WIK16d] *Wikipedia: HSV-Farbraum*. https://de.wikipedia.org/wiki/HSV-Farbraum. Version: 03 2016

[WIK16e] *Wikipedia: Liste*. https://de.wikipedia.org/wiki/Liste_(Datenstruktur). Version: 04 2016

[WIK16f] *Wikipedia: Modelltransformation*. http://de.wikipedia.org/wiki/Modelltransformation. Version: 03 2016

[WIK16g] *Wikipedia: Nachbarschaft (Bildverarbeitung)*. https://de.wikipedia.org/wiki/Nachbarschaft_(Bildverarbeitung). Version: 04 2016

[WIK16h] *Wikipedia: Otsu's method*. https://en.wikipedia.org/wiki/Otsu%27s_method. Version: 04 2016

[WIK16i] *Wikipedia: Rastergrafik*. https://de.wikipedia.org/wiki/Rastergrafik. Version: 03 2016

[WIK16j] *Wikipedia: RGB-Farbraum*. https://de.wikipedia.org/wiki/RGB-Farbraum. Version: 03 2016

[WIK16k] *Wikipedia: Schwellwertverfahren*. https://de.wikipedia.org/wiki/Schwellenwertverfahren. Version: 04 2016

[WIK16l] *Wikipedia: Vektorgrafik*. https://de.wikipedia.org/wiki/Vektorgrafik. Version: 03 2016