Beispiel 3 - Kamerasensoren

VU Einführung in Visual Computing, TU Wien, SS 2023 14.03.2023

1 Übersicht

Die Sensoren von handelsüblichen digitalen Kameras erzeugen nicht direkt Farbbilder. Jeder Bildpunkt reagiert nur auf eine der drei Farbkomponenten des einfallenden Lichts (Rot, Grün oder Blau). Es wird also ein Graustufenbild aufgezeichnet, aus welchem die Farben erst später durch die Firmware der Kamera berechnet werden. In diesem Beispiel soll diese Grundverarbeitung in der Kamera, die normalerweise verborgen bleibt, in Python umgesetzt werden.

1.1 Bewertung

Vorverarbeitung:	7 Punkte
Demosaicing:	12 Punkte
Weißabgleich:	7 Punkte
Gammakorrektur:	12 Punkte
Kontraststärkung:	7 Punkte
Gesamt:	**45 Punkte**

1.2 Allgemeine Informationen zur Abgabe

Bitte stellt sicher, dass Python beim Ausführen der abgegebenen Dateien keine Fehlermeldungen ausgibt, das Skript abstürzt oder ähnliches. **Wir können keinen fehlerhaften bzw. nicht ausführbaren Code bewerten!**

Es liegt in deiner Verantwortung rechtzeitig **vor der Deadline** zu kontrollieren, ob deine Abgabe funktioniert hat. Sollte der Upload deiner Dateien aus irgendeinem Grund nicht funktionieren, poste im TUWEL-Forum (sofern das Problem aktuell noch kein anderer Student gemeldet hat) und schreib eine E-Mail an **evc@cg.tuwien.ac.at**, damit wir schnellstmöglich darauf reagieren können.

Kontrolliere deine Abgabe:

- Werden die hochgeladenen Dateien angezeigt?
- Kannst du die Dateien herunterladen und öffnen oder im Browser anzeigen?
- Hast du die richtigen Dateien abgegeben?

Abzugeben ist eine ZIP-Datei, die folgende Dateien beinhaltet:

- evc_black_level.py
- evc_demosaic.py
- evc_white_balance.py
- evc_gamma_correction.py
- evc_compute_binary.py
- evc_histogram_clipping.py

1.3 Allgemeine Hinweise

Die gesamte Aufgabe wird innerhalb des Jupyter-Notebooks namens `TASK3.ipynb` behandelt. Einzelne Schritte werden zur erleichterten Bedienung mit Hilfe einer Grafischen Benutzeroberfläche ausgeführt.

Zwischenergebnisse: Die Zwischenergebnisse dieser Aufgabe können teilweise ein schwarzes Bild darstellen, solange einzelne Teile noch nicht implementiert sind. Wenn alles korrekt implementiert ist, sollte ein Bild erkenntlich sein.

2 Datensatz

Lade zuerst die Angabe herunter, welche das Python-Framework beinhaltet! Die Bilder sind im TUWEL-Kurs separat herunterzuladen, diese wurden mit einer Spiegelreflexkamera mit 16-Bit Farbtiefe aufgenommen, die Auflösung wurde aber für die Übung reduziert. Alle Bilder sind durch das Urheberrecht geschützt und dürfen nur zum Zweck dieser LVA verwendet werden.

In den weiteren Aufgaben dieses Beispiels sollen alle Funktionen, welche im Framework deklariert sind, implementiert werden. **Änderungen sind nur in den Dateien mit dem Präfix *evc_* erlaubt**, der Rest gehört zum User-Interface und darf nicht verändert werden. Zusätzlich zu den Angaben der einzelnen Unterbeispiele finden sich in den jeweiligen Dateien auch detaillierte Anweisungen zum Code.

Die Signaturen (Name, Eingabe, Ausgabe) der vordefinierten Funktionen dürfen in diesem Beispiel nicht verändert werden.

3 Vorverarbeitung

Theorie in Vorlesung: *Point Operations*

Die Methodensignatur darf auf keinen Fall verändert werden!
Wenn nicht anders angegeben, dürfen keine Schleifen oder Rekursionen verwendet werden!

Alle Bilder sind als 16-Bit TIFFs gespeichert. Neben den Helligkeitsinformationen sind in den Dateien Metadaten für jedes Bild gespeichert. Für diese Übung werden der Schwarzwert (*BlackLevel*) und der neutrale Weißabgleich (*AsShotNeutral*) benötigt, welche von der Kamera bei der Aufnahme bestimmt wurden. Diese Metadaten können mit Hilfe der Python-Bibliothek `Pillow` abgefragt werden. Generiere beispielsweise das Metadaten-Dictionary mittels

```
meta_dict = {TAGS[key] : img.tag[key] for key in img.tag_v2}
```

und extrahiere die Informationen mittels den Keys *BlackLevel* und *AsShotNeutral*.

Physikalische Eigenschaften der Kamerasensoren verursachen Rauschen, welches vor allem in dunklen Regionen zu beobachten ist. Deshalb wird ein Durchschnitt des Rauschens bei der Aufnahme ermittelt und gespeichert. Vor dem Demosaicing muss dieser Wert von der Intensität jedes Bildpunktes abgezogen werden, damit das Rauschen unterdrückt wird.

3.1 evc_black_level.py

Es sind 2 Funktionen zu implementieren. Diese Funktionen werden von der Hauptfunktion der Datei *evc_black_level* in der richtigen Reihenfolge aufgerufen:

- *evc_read_file_info*
- *evc_transform_colors*

ACHTUNG: Die Hauptfunktion darf dabei nicht verändert werden!
ACHTUNG: Die Funktionen müssen genau die beschriebene Funktion erfüllen!

3.2 Funktion *evc_read_file_info*

Diese Funktion liest den Schwarzwert (*blackLevel*) und den neutralen Weißabgleich (*asShotNeutral*) aus der Datei, die durch *filename* beschrieben ist, aus.

3.3 Funktion *evc_transform_colors*

Diese Funktion verschiebt und skaliert den Kontrast so, dass Schwarz (*blackLevel* und Werte darunter) auf 0 und Weiß auf 1 abgebildet werden. Dazu soll das Eingabebild vom Intervall [*blackLevel*, 65535]

auf das Intervall [0.0, 1.0] abgebildet werden.

HINWEIS: Beachte, dass der Input zuerst auf den Datentyp *float* konvertiert werden muss.

In anderen Worten: Der Schwarzwert soll 0.0 und der Wert 65535 (Weiß) soll 1.0 werden (siehe Abbildung 1). Alle Werte zwischen BlackLevel und 65535, sollten zwischen 0.0 und 1.0 liegen und alle Werte kleiner dem Schwarzwert müssen auch auf 0.0 gesetzt werden.

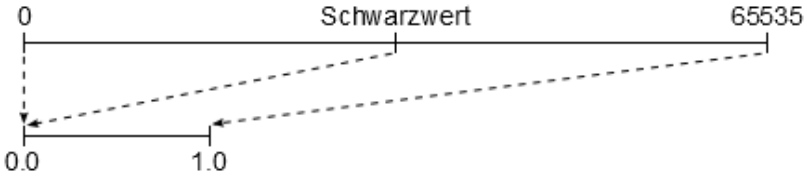


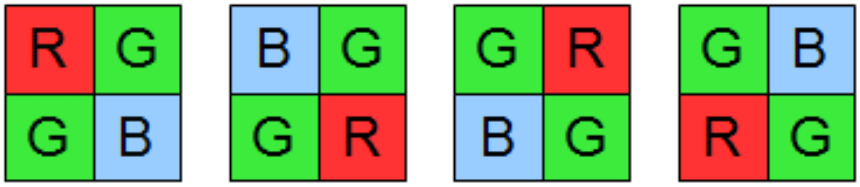
Abbildung 1: Das Intervall [Schwarzwert, 65535] soll auf den Bereich [0,1] abgebildet werden. Alle Werte kleiner dem Schwarzwert sollen danach 0 sein.

4 Demosaicing

Theorie in Vorlesung: *Bildaufnahme*

Die Methodensignatur darf auf keinen Fall verändert werden!
Wenn nicht anders angegeben, dürfen keine Schleifen oder Rekursionen verwendet werden!

Der Bildsensor ist mit einem Farbfilter (Bayer Pixel Pattern) überzogen, der nur eine bestimmte Farbkomponente zum Sensor durchlässt. Diese Filter sind meistens in 2×2 Pixelmustern angeordnet. Da die Sensitivität des menschlichen Auges im grünen Bereich am besten ist, wird der grüne Filter auf zwei der vier Sensoren aufgetragen. Verschiedene Kameramodelle verwenden verschiedene Muster, hauptsächlich sind aber die folgenden vier in Verwendung (siehe Abbildung 2): RGGB, BGGR, GRBG, GBRG. Im digitalisierten Bild entspricht jeder Sensor einem Pixel.



a) RGGB b) BGGR c) GRBG d) GBRG

Abbildung 2: In diesem Beispiel gibt es 4 mögliche Bayer Filter: RGGB, BGGR, GRBG, GBRG.

In einem Bild, welches mit dem RGGB Filter aufgezeichnet wurde, sind die Farbwerte beispielsweise wie in Abbildung 3 verteilt.

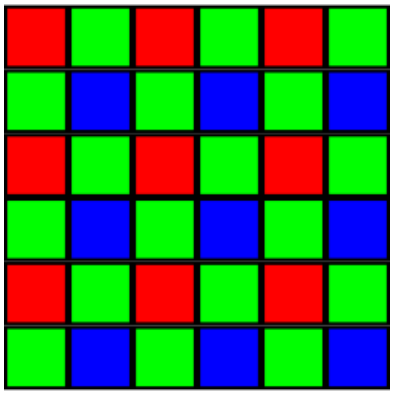


Abbildung 3: Verteilung mittels RGGB Filter.

Um alle drei Farbkanäle für jedes Pixel zu bekommen, müssen an den Stellen, an denen Informationen fehlen, die benachbarten Werte interpoliert werden.

Für fehlende Blauwerte funktioniert dies wie in Abbildung 4 dargestellt.

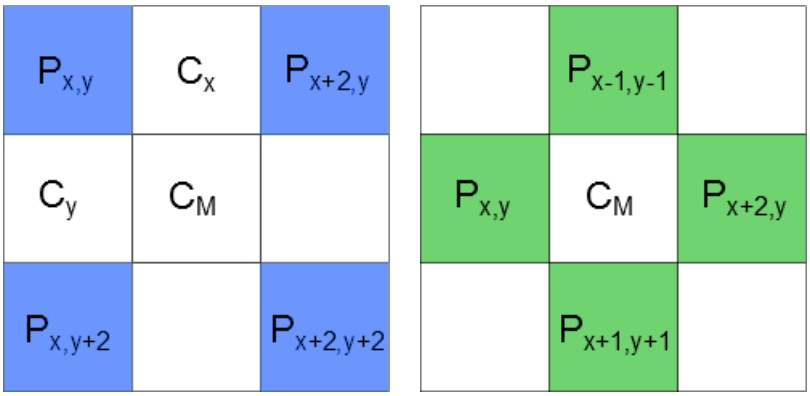


Abbildung 4: Illustration benachbarter blauer bzw. grüner Bildpunkte.

Betrachtet man den ersten blauen Bildpunkt (Pixel) $P_{x,y}$ sind dessen direkte Nachbarn C_x, C_y und C_M. Die nächstgelegenen blauen Bildpunkte sind $P_{x+2,y}$, $P_{x,y+2}$ und $P_{x+2,y+2}$ (siehe Abbildung 4). Um C_x, C_y und C_M zu berechnen, wird das Mittel der angrenzenden 2 oder 4 Blauwerte berechnet (auch diagonal). Es ist daher stets die komplette Achter-Nachbarschaft (Engl. 8-connected neighbourhood) zu berücksichtigen.

4.1 evc_demosaic.py

Es sind 4 Funktionen zu implementieren. Diese Funktionen werden von der Hauptfunktion der Datei *evc_demosaic* in der richtigen Reihenfolge aufgerufen:

- evc_demosaic_pattern
- evc_transform_neutral
- evc_interpolate
- evc_concat

ACHTUNG: Die Hauptfunktion darf dabei nicht verändert werden!
ACHTUNG: Die Funktionen müssen genau die beschriebene Funktion erfüllen!

4.2 Funktion evc_demosaic_pattern

Diese Funktion soll mit Hilfe des richtigen Patterns die Rot-, Grün- und Blauwerte aus dem Input-Bild auslesen und zurückliefern.

Um dies effizient in Python zu implementieren, erzeugt man drei Matrizen mit der Größe des Eingabebildes (`np.zeros(img.shape)`) für die drei Farbkanäle Rot, Grün und Blau und kopiert anschließend die Werte für Rot, Grün und Blau in die jeweilige Matrix. Dabei kommt es darauf an, welches Bayer-Pattern in deinem Datensatz verwendet wurde (alle Testbilder verwenden das gleiche Pattern). Implementiere den Algorithmus für ein Pattern und teste es an den Testbildern.

Sollten die Farben in einem Testbild nicht stimmen, musst du den Code anpassen (**HINWEIS:** Wir empfehlen alle 4 Bayer Pixel Patterns zu testen!). Beispielsweise muss für das RGGB Pattern in jeder zweiten Zeile jedes zweite Pixel im Rot-Kanal gespeichert werden:

```
R[::2, ::2] = img[::2, ::2]
```

Dieselbe Vorgehensweise kann man auch für die Grün- und Blauwerte verwenden. Beachte hierbei wieder die etwas andere Verteilung der Grünwerte.

4.3 Funktion evc_transform_neutral

Diese Funktion soll den neutralen Weißabgleich durchführen. Dabei wird der Rotwert des Bildes durch den Rotwert des neutralen Weißwertes dividiert (analog für Grün und Blau). Das Ergebnis soll dann zurückgegeben werden (wieder in RGB gespeichert werden).

4.4 Funktion evc_interpolate

Diese Funktion soll die Werte in jedem Kanal nun interpolieren, sodass die Lücken gefüllt werden. Dies kann man mit der Funktion *imfilter* bewerkstelligen, indem man einen geeigneten Filter-Kernel verwendet. Um fehlende Grünwerte zu berechnen, verwendet man 1/4 des linken, rechten, oberen und unteren Pixels. Der Filter sieht daher wie folgt aus:

$$\text{filter} = \begin{bmatrix} 0.00 & 0.25 & 0.00 \\ 0.25 & 1.00 & 0.25 \\ 0.00 & 0.25 & 0.00 \end{bmatrix} \tag{1}$$

Für Rot und Blau benötigt man einen etwas komplexeren Filter, den du dir mit einem kleinen Beispiel auf dem Papier schnell herleiten kannst.

HINWEIS: Es handelt sich wieder um einen 3×3 Filter.
Pythonen: `scipy.ndimage.correlate(a, filter, mode='constant')`

4.5 Funktion evc_concat

Diese Funktion verbindet die drei Farbkomponenten zu einem Bild. Hierzu ist es üblich, dass nach dem Verbinden die resultierende Matrix die Dimensionen Zeilen $\times$ Spalten $\times$ Farbkanaele besitzt.

Python: `np.stack`

Die Aufgaben für dieses Beispiel lassen sich wie folgt zusammenfassen: Finde heraus, welches Bayer-Muster im Datensatz verwendet wird und implementiere eine Methode für lineare Interpolation wie

eben beschrieben in der Datei *evc_demosaic.py*. Weitere Hilfe ist in den Unterlagen zur Vorstellung der Beispiele zu finden.

5 Weißabgleich

Theorie in Vorlesung: *Bildaufnahme*

Die Methodensignatur darf auf keinen Fall verändert werden!
Wenn nicht anders angegeben, dürfen keine Schleifen oder Rekursionen verwendet werden!

Ein Bildpunkt, bei dem alle drei Farbkomponenten gleich sind, hat keine Farbe und ist Grau (im Extremfall Schwarz oder Weiß). Wenn man also von einem Weißabgleich spricht, handelt es sich eigentlich um einen Grauabgleich. Der Bayer Filter hat eine ungleichmäßige Sensitivität auf die drei Grundfarben, daher sind Bildpunkte, welche Grau sein sollten, anders eingefärbt. Die Kamera versucht automatisch, die Farben abzustimmen. Der abgeschätzte Weißpunkt wurde in der vorigen Aufgabe für den **neutralen Weißabgleich** verwendet. Oft wird nachträglich noch ein manueller Ausgleich benötigt.

Dazu soll der User einen Bildpunkt selektieren, von dem er annimmt, dass er Weiß sei. Nach dem Abgleich wird dieser selektierte Bildpunkt den Farbwert $(1, 1, 1)$ bekommen. Hellere Punkte werden Intensitäten über 1 aufweisen. Bei Bildpunkten, die nahe an der maximalen Intensität liegen, können sogar falsche Farben entstehen. Das Histogramm-Clipping in der letzten Aufgabe wird diese Probleme lösen. Beachte, dass der selektierte Bildpunkt auch 0-Werte enthalten kann.

5.1 GUI

Wähle einen Bildpunkt (dies ist in beiden Bildern möglich), dessen Farbe als gewünschtes Weiß für den Weißabgleich genommen wird. Links ist das Originalbild, rechts wird das Ausgabebild angezeigt. Mit `Reset` kann der Weißabgleich resetted werden, mit `OK` wird der aktuelle Bildpunkt gespeichert und das abgeglichene Bild zurückgegeben.

5.2 evc_white_balance.py

Nachdem die Farbe des Referenzpunktes bestimmt wurde, skaliert man alle Bildpunkte jedes Farbkanals mit dem Kehrwert dieser Farbe. Durch diese Normalisierung bekommen graue Bildpunkte gleiche Anteile aller Farbkomponenten. Hierbei ist darauf zu achten, dass der Referenzpunkt auch 0-Werte enthalten kann.

6 Gammakorrektur

Theorie in Vorlesung: *Point Operations*

Die Methodensignatur darf auf keinen Fall verändert werden!
Wenn nicht anders angegeben, dürfen keine Schleifen oder Rekursionen verwendet werden!

Das menschliche Auge reagiert auf Variationen in dunklen Bereichen empfindlicher als in hellen. Unser Sehempfinden ist somit nicht linear. Moderne Bildschirme ahmen diese Eigenschaft nach. Um die linear abgetasteten Intensitätswerte für die Wahrnehmung anzupassen, wird das Bild mit einer Potenzfunktion mit dem Exponenten y^{-1} korrigiert. Eine gute und verbreitete Wahl ist $y = 2.2$.

6.1 GUI

Der Schieber in der GUI steuert den y-Wert, Bild und Histogramm passen sich interaktiv an. Die Kontrollbox mit dem Text `Keep color balance` schaltet zwischen zwei Korrekturverfahren um.

6.2 evc_gamma_correction.py

Es sollen zwei Arten der Gammakorrektur implementiert werden. Die einfache Variante potenziert die Farbkanäle unabhängig voneinander. Das Eingabebild wird also nur mit y^{-1} potenziert. Dadurch wird der vorige Weißabgleich zunichte gemacht, da sich die Anteile der Farbkomponenten nicht gleichmäßig verändern. Ist y gleich 0, würde dies eine Division durch 0 zur Folge haben. Um dies zu verhindern, soll y automatisch auf 0.0000000001 gesetzt werden, wenn es kleiner als 0.0000000001 ist. Die zweite Variante korrigiert nur die Helligkeit, während die Farbanteile erhalten bleiben.

Es sind vier Funktionen zu implementieren. Diese Funktionen werden von der Hauptfunktion der Datei `evc_gamma_correction` in der richtigen Reihenfolge aufgerufen:

- evc_compute_brightness
- evc_compute_chromaticity
- evc_gamma_correct
- evc_reconstruct

6.3 Funktion evc_compute_brightness

Zuerst berechnet man mit der in `evc_gamma_correction.py` zur Verfügung gestellten Funktion `rgb2gray` die Helligkeitswerte des Bildes. Da das Ergebnis von `rgb2gray` im Intervall [0, 1] abgeschnitten wird, bleiben jedoch die Intensitäten der Pixel nicht erhalten (zu helle Werte werden auf 1 abgebildet). Daher berechnet man sich, bevor man `rgb2gray` anwendet, den maximalen Wert aller drei Farbkanäle mit

`np.max`, normalisiert das Bild manuell, indem man es mit dem Kehrwert des maximalen Wertes der Farbkanäle multipliziert, wendet `rgb2gray` auf das normalisierte Bild an und multipliziert das Resultat letztendlich wieder mit dem maximalen Wert der Farbkanäle.

6.4 Funktion evc_compute_chromaticity

Weiters benötigen wir die Chromatizität. Sie beschreibt eine Farbe ohne den Helligkeitswert. Dadurch ist es möglich, die Helligkeitswerte getrennt von der Farbe zu verändern und später wieder zusammenzuführen. Die Chromatizität berechnet man, indem die Verhältnisse der Farbkanäle zur Helligkeit berechnet werden (die Farbkanäle des Input-Bildes werden durch die Helligkeit dividiert).

Python: `np.dstack`

6.5 Funktion evc_gamma_correct

Nun führt man die Gammakorrektur wie bei der ersten Methode durch, nur dass in diesem Falle die berechneten Helligkeitswerte korrigiert werden.

HINWEIS: Sowohl für die einfache als auch für diese Variante kann dieselbe Funktion verwendet werden, d.h. hier ändert sich lediglich das Bild, auf das die Korrektur angewendet wird!

6.6 Funktion evc_reconstruct

Die Rekonstruktion der Farbwerte erfolgt durch Multiplikation der korrigierten Helligkeit mit der Chromatizität.

Python: `np.dstack`

7 Kontrastverstärkung

Theorie in Vorlesung: *Point Operations*

Die Methodensignatur darf auf keinen Fall verändert werden!
Wenn nicht anders angegeben, dürfen keine Schleifen oder Rekursionen verwendet werden!

Nach der Gammakorrektur werden die dunkelsten Bildpunkte heller und so entsteht oft der Eindruck von niedrigem Kontrast. Am anderen Ende des Spektrums gibt es immer noch Werte größer als 1.

Die Kontrastverstärkung ändert die Intensitäten so, dass die dunkelsten Bereiche auf 0 abgebildet werden und die hellsten auf 1. Der Rest der Intensitäten kann dann auf das ganze Spektrum ausgedehnt werden, wodurch der Kontrast gestärkt wird.

Die weißen, überbelichteten Regionen haben nach dem Weißabgleich meistens ihre neutrale Farbe verloren. Das kann behoben werden, indem das Histogramm so abgeschnitten wird, dass alle drei Farbkomponenten der überbelichteten Bildpunkte auf 1 abgebildet werden.

7.1 GUI

Es sollen zwei Punkte innerhalb des Histogramms ausgewählt werden (`Low` und `High`), dafür sind zwei Felder vorgegeben. Zur Orientierung wird das Originalhistogramm des Bildes angezeigt. `Low` bestimmt die untere und `High` die obere Grenze, an der das Histogramm abgeschnitten wird. Alle Bildpunkte, die schwarz sind, bleiben erhalten, alle mit Farbe werden abgeschnitten. Wenn ein Bildpunkt im Binärbild nur rot ist, heißt das, dass nur die rote Komponente abgeschnitten wird, bei Gelb wird die rote und grüne Komponente abgeschnitten, usw.

7.2 evc_compute_binary.py

In dieser Datei soll ein Binärbild berechnet werden, welches für die Visualisierung der Histogrammgrenzen im Interface verwendet wird. D.h. alle Werte kleiner oder gleich einem gewissen Schwellwert müssen 0 und alle Werte darüber 1 werden. Je nachdem, ob die untere oder die obere Grenze selektiert wird (gegeben durch den Parameter `top`), soll das Ergebnis invertiert werden. Verwende die logische Indizierung, um das Bild zu binarisieren. Das zurückgegebene Binärbild soll vom Typ *float* sein.

7.3 evc_histogram_clipping.py

Die Intensitätswerte sollen linear auf [0, 1] abgebildet werden. Dabei werden alle Werte, die dunkler als die untere Schranke sind, auf 0 und alle heller als die obere Schranke auf 1 abgebildet. Das Verfahren ähnelt der Aufgabe mit dem Schwarzwert gleich nach dem Einlesen des Bildes. Es sind drei Funktionen zu implementieren.Diese Funktionen werden von der Hauptfunktion der Datei `evc_histogram_clipping` in der richtigen Reihenfolge aufgerufen:

- *evc_prepare_histogram_range*
- *evc_transform_histogram*
- *evc_clip_histogram*

ACHTUNG: Die Hauptfunktion darf dabei nicht verändert werden!
ACHTUNG: Die Funktionen müssen genau die beschriebene Funktion erfüllen!

7.4 Funktion evc_prepare_histogram_range

Diese Funktion bestimmt zunächst die für das Normalisieren notwendige obere und untere Schranke, die bei der Normalisierung auf [0,1] abgebildet werden sollen. Falls $low < 0$ ist, muss es auf 0 gesetzt werden. Falls $high >$ als die maximale Intensität des Bildes ist, muss es auf die maximale Intensität gesetzt werden. Die Ergebnisse sollen anschließend in *newLow* und *newHigh* gespeichert werden.

7.5 Funktion evc_transform_histogram

Diese Funktion führt die "Histogram Normalization" durch und bildet das Intervall [*newLow*, *newHigh*] auf [0, 1] ab.

HINWEIS: In dem Fall, dass der aktuelle Weißpunkt kleiner als die maximale Intensität des Bildes ist, entstehen hier Werte größer 1. Dieses Problem wird durch die Funktion *evc_clip_histogram* gelöst.

7.6 Funktion evc_clip_histogram

Diese Funktion setzt alle Werte, die nach der Transformation des Histogramms noch < 0 sind, auf 0 und Werte, die > 1 sind, auf 1.