

(a) Eingabe (rot).

(b) Eingabe (grün).

(c) Eingabe (blau).

(d) Ausgabe.

Abbildung 1: Beispiel einer kolorierten Fotografie (Image Colorization). Drei einzelne Farbkanäle (rot, grün, blau) werden zu einem Farbbild kombiniert.

# CV/task1 — Image Colorization

## 1 Überblick

Ziel dieser Aufgabenstellung ist es, grundlegende Funktionalitäten sowie einen Teil des erweiterten Funktionsumfangs der OpenCV-Library kennenzulernen. Dafür soll Schritt für Schritt ein automatisiertes *Image Colorization*-Programm entwickelt werden, bei dem aus drei Einzelbildern (Farbkanäle rot, grün und blau) ein koloriertes Farbbild erzeugt werden soll. Ein Beispiel dazu finden Sie in Abbildung 1.

Für die ausreichende Darstellung eines Farbbildes sind zumindest drei Farbinformationen pro Pixel notwendig (z. B. RGB, Lab, HSV, ...). Diese werden als Koordinaten in dem jeweiligen Farbraum interpretiert und definieren dadurch eindeutig die Farbe des Pixels. Eine gängige Form der Darstellung bietet der RGB-Farbraum mit seinen drei Koordinaten für den (roten) R-, (grünen) G- und (blauen) B-Kanal. Dieses Beispiel befasst sich mit der Rekonstruktion eines Farbbildes aus drei separaten, nacheinander aufgenommenen und teils verschobenen Intensitätsbildern. Diese wurden nach dem Verfahren von Sergei Michailowitch Prokudin-Gorski<sup>1</sup> erstellt und entstanden bereits Anfang des 20.Jahrhunderts. Dabei wurden in kurzem zeitlichen Abstand drei Intensitätsbilder durch jeweils einen roten, blauen und einen grünen Farbfilter aufgenommen. Um zu einem späteren Zeitpunkt ein Farbbild wiedergeben zu können, wurden drei Projektoren – einer mit einem roten, einer mit einem grünen und einer mit einem blauen Farbfilter – verwendet, mit denen die Einzelbilder übereinander projiziert wurden.

---

<sup>1</sup>[https://de.wikipedia.org/wiki/Sergei\\_Michailowitsch\\_Prokudin-Gorski#Fotografisches\\_Verfahren](https://de.wikipedia.org/wiki/Sergei_Michailowitsch_Prokudin-Gorski#Fotografisches_Verfahren)

## 2 Aufgaben

Das zur Verfügung gestellte Framework beinhaltet ein main.cpp (sollte nicht bearbeitet werden, dient zur automatischen Ausführung) und ein algorithm.cpp, in dem die Aufgaben in den jeweiligen Funktionen (compute\_gradient, compute\_binary, edge\_detection, translate\_img, edge\_matching, combine\_images und crop\_image) umgesetzt werden sollen.

Die benötigten Parameter für die Aufgaben sind in den jeweiligen JSON-Dateien hinterlegt. Die Eingabedaten werden aus den JSON-Dateien bereits automatisch eingelesen und können sofort zur Berechnung herangezogen werden. Der Inhalt dieser Aufgabe beschränkt sich einzlig und allein auf die Image Colorization-Funktionen und nicht auf etwaigen Aufwand, der sich durch die Programmiersprache ergibt. Das Framework ist so aufgebaut, dass Sie die relevanten Funktionen implementieren müssen um die gewünschte Ausgabe zu erreichen.

Das Beispiel ist in mehrere Unteraufgaben gegliedert. Der Ablauf besteht aus den folgenden Schritten:

- Kantendetektion (1 Punkt)
  - Gradienten-Berechnung (1.5 Punkte)
  - Binärbild-Berechnung (1.5 Punkte)
- Translation eines Bildes (3 Punkte)
- Berechnung und Durchführung der Verschiebung (5 Punkte)
- Kombinieren der Farbkanäle (2 Punkte)
- Zuschneiden des Resultates (2 Punkte)
- Bonus (3 Punkte)

**Diese Aufgabe ist mit Hilfe von OpenCV<sup>2</sup> 4.2.0 zu implementieren. Nutzen Sie die Funktionen, die Ihnen OpenCV zur Verfügung stellt und achten Sie auf die unterschiedlichen Parameter und Bildtypen<sup>3</sup>.**

---

<sup>2</sup><http://opencv.org/>

<sup>3</sup>OpenCV nutzt aus historischen Gründen eigentlich das BGR-Farbformat statt des RGB-Farbformats, d.h. die Reihenfolge der Kanäle ist vertauscht. Berücksichtigen Sie dies in Ihrer Implementation.

## 2.1 Gradienten-Berechnung (1.5 Punkte)



(a) Eingabe (R-Kanal).



(b) Ausgabe (Gradient R-Kanal).

Abbildung 2: Eingabe und Ausgabe der Gradienten-Berechnung.

Diese Aufgabe soll in der Funktion `compute_gradient(...)` gelöst werden. Die Gradienten-Berechnung soll mit dem Sobel-Operator erfolgen. Die hierfür anwendbare OpenCV Funktion `cv::Sobel` detektiert gerichtete Änderungen unter Zuhilfenahme von Ableitungen. Ein großer Gradientenwert in gewissen Bildregionen deutet auf stark ausgeprägte Intensitätsveränderungen – verursacht durch Kanten, Farbübergänge, Belichtungsunterschiede, etc. – in diesen hin. Für die Ableitung in x-Richtung  $G_x$  wird das Bild  $I$  mit dem Kernel  $S_x$  gefaltet. Analog wird für die Ableitung in y-Richtung  $G_y$  das Bild  $I$  mit dem Kernel  $S_y$  gefaltet, sodass sich die Formeln

$$G_x = S_x * I = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} * I \quad (1)$$

und

$$G_y = S_y * I = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix} * I \quad (2)$$

ergeben. Mit  $*$  wird hierbei die Faltungsoperation bezeichnet. Bestimmen Sie zuerst die jeweils *erste* Ableitung in die Richtungen  $x$  und  $y$ . Wählen Sie für den Parameter der die Bit-Tiefe des Output-Bildes beschreibt den Typ `CV_32F`. Kombinieren Sie danach die beiden Gradientenbilder gemäß der Formel

$$G = \sqrt{G_x^2 + G_y^2}, \quad (3)$$

wobei  $G$  das Gesamtgradientenbild bezeichnet. Die Funktion `compute_gradient` wird dann in Aufgabe 2.3 verwendet.

### Nützliche Funktionen:

- `cv::add(...)`
- `cv::pow(...)`
- `cv::Sobel(...)`
- `cv::sqrt(...)`

## 2.2 Binärbild-Berechnung (1.5 Punkte)

Diese Aufgabe soll in der Funktion `compute_binary(...)` umgesetzt werden. In dieser Aufgabe soll das zuvor berechnete Gradientenbild in ein Intensitätsbild mit dem Datentyp `CV_8UC1` konvertiert und eine geeignete Skalierung verwendet werden. Nutzen Sie hierfür die Methode `Mat::convertTo(...)` und skalieren Sie die Pixelwerte, sodass die Werte von  $p_{min}$  – welcher hier dem niedrigsten im Gradientenbild vorkommenden Pixelwert entspricht – bis  $p_{max}$  – welcher dem größten Pixelwert entspricht – in das Intervall  $[0, 255]$  abgebildet werden. Dieser Schritt kann in mathematischer Schreibweise folgendermaßen dargestellt werden:

$$y = (x - p_{min}) \cdot \frac{255}{p_{max} - p_{min}}, \quad (4)$$

wobei  $y$  den Intensitätswert im Intervall  $[0, 255]$  darstellt, welcher mit dem Wert  $x$  aus dem Ursprungsintervall korrespondiert. Um ein besseres Ergebnis der Kantendetektion zu erhalten und fälschlich detektierte Kanten zu eliminieren, wird nun Thresholding auf das Bild angewandt, sodass alle Werte, die kleiner oder gleich dem Parameter  $\tau$  sind, 0 gesetzt werden, und jene die größer als der Parameter sind, werden auf 255 gesetzt. Das binäre Bild  $E$

$$E(x, y) = \begin{cases} 255, & \text{wenn } G(x, y) > \tau \\ 0, & \text{sonst} \end{cases} \quad (5)$$

enthält somit nur noch die Kanteninformation. Der Wert für  $\tau$  wird in dem Parameter `edge_threshold` übergeben. Das Resultat  $E$  (siehe Abbildung 3) soll in die Matrix `output_image` gespeichert werden. Die Funktion `compute_binary` wird anschließend in der Aufgabe 2.3 verwendet.

### Nützliche Funktionen:

- `Mat::convertTo(...)`
- `Mat::minMaxLoc(...)`

- `Mat::threshold(...)`

### Verbotene Funktionen:

- `cv::normalize(...)`

## 2.3 Kantendetektion (1 Punkte)

Diese Aufgabe soll in der Funktion `edge_detection(...)` gelöst werden. Nun sollen für den jeweiligen Farbkanal (B, G und R) die Kanten detektiert werden. Hierzu sollen die zuvor implementierten Funktionen zur Berechnung der Gradienten `compute_gradient(...)` und Binärbilderstellung `compute_binary(...)` aufgerufen werden. Die Ergebnisbilder der Kantendetektion sollen in den Vector `img_edges_BGR` in der gegebenen Reihenfolge abgespeichert werden (nämlich das Bild für den Blaukanal zuerst, danach das Bild für den Grünkanal und am Schluss das Bild für den Rotkanal). Das Ergebnis der Kantendetektion wird in Abbildung 3 gezeigt.



(a) Eingabe (G-Kanal).



(b) `img_g_edge` (Kanten G-Kanal).

Abbildung 3: Eingabe und Ausgabe der Kantendetektion.

## 2.4 Translation eines Bildes (3 Punkte)

Diese Aufgabe soll in der Funktion `translate_img(...)` implementiert werden. In dieser Aufgabe soll ein Eingangsbild um einen gewissen Offset in Richtung der Reihen (`r_offset`) und der Spalten (`c_offset`) verschoben werden. Hierzu muss das gesamte Bild (Reihen und Spalten) durch iteriert werden und der durch den Offset berechnete Pixel in das Ausgangsbild gespeichert werden. Das Bild soll hierbei bei einem positiven Offset in Richtung



(a) Eingabe (R-Kanal).



(b) Ausgabe (R-Kanal verschoben).

Abbildung 4: Eingabe und Ausgabe der Translation.

der Reihen (`r_offset`) und der Spalten (`c_offset`) nach rechts bzw. unten verschoben werden (siehe Abbildung 4). Die Pixel in dem dabei entstandene Bereich links und oberhalb des verschobenen Bildes sollen auf 0 gesetzt werden. Das resultierende Ausgangsbild sollte die gleiche Größe wie das Eingangsbild haben und in die Matrix `output_image` gespeichert werden. Die Funktion `translate_img(...)` soll dann in Aufgabe 2.5 zum Einsatz kommen und für die Translation einzelner Channels verwendet werden.

#### Nützliche Funktionen:

- Zugriff auf einzelnen Pixel in einem Grauwertbild: `Mat::at<uchar>(y, x)`

#### Verbotene Funktionen:

- `cv::warpAffine(...)`

## 2.5 Berechnung und Durchführung der Verschiebung (5 Punkte)

Diese Aufgabe soll in der Funktion `edge_matching(...)` umgesetzt werden. Da die unterschiedlichen Eingabebilder leichte Abweichungen bezüglich der Position der in ihnen vorkommenden Objekte aufweisen, müssen in diesem Schritt die Verschiebungen (jeweils in  $x$ - und  $y$ -Richtung) zwischen den Bildern bestimmt werden. Dabei wird das Eingabebild des roten Farbkanals als Referenz gewählt und die Channels G und B werden so verschoben, dass ihre Kanten mit denen von R übereinstimmen. Hierfür wird innerhalb eines durch den Parameter `match_window_size` definierten Bereiches die optimale Verschiebung berechnet. Verwenden Sie für die Translation die zuvor generierte Funk-



(a) Eingabe (G-Kanal).



(b) Ausgabe (G-Kanal verschoben).

Abbildung 5: Eingabe und Ausgabe der Verschiebungsbestimmung und Translation der Bilder.

tion `translate_img(...)`. Um diese optimale Verschiebung zu bestimmen, werden die Kantenbilder von B und G innerhalb des Intervalls

$$[r_{min}, r_{max}] = \left[ -\left\lfloor \frac{\text{match\_window\_size}}{2} \right\rfloor, \left\lceil \frac{\text{match\_window\_size}}{2} \right\rceil \right] \quad (6)$$

verschoben und eine Punktzahl für die jeweilige Verschiebung bestimmt. Diese Punktzahl ist dabei definiert als die Summe der Kantenpixel im Referenzkanal, welche in dem transformierten Bild ebenfalls auf einem Kantenpixel liegen. Gleichung 7 veranschaulicht diesen Zusammenhang:

$$S(i, j) = \sum_{x, y} \delta(E(x, y), E'(x + i, y + j)) \quad \text{mit} \quad \delta(a, b) = \begin{cases} 1, & \text{wenn } a = b = 255 \\ 0, & \text{sonst} \end{cases}. \quad (7)$$

$E$  ist das Kantenbild des Referenzkanals und  $E'$  eines der Kantenbilder der Kanäle B und G. Die Variablen  $x$  und  $y$  sind die Koordinaten für alle Pixel im Bild. Es soll jene Verschiebung  $(i, j)$  der Kantenbilder der Kanäle B und G gefunden werden, welche die Score-Funktion  $S(i, j)$  maximiert:

$$(i, j)_{max} = \underset{i, j}{\operatorname{argmax}} S(i, j) \quad \text{mit} \quad i, j \in [r_{min}, r_{max}]. \quad (8)$$

Wie in Abbildung 6 zu sehen ist, wird das zu verschiebende Bild (gelb) zunächst um die maximale Verschiebung ( $i = r_{min}$  und  $j = r_{min}$ ) verschoben. Danach wird die Punktzahl für diese Verschiebung bestimmt, mit der Punktzahl der bisher besten Verschiebung verglichen (und evtl. gespeichert), danach wird die x- oder y-Verschiebung angepasst und wiederum die Punktzahl bestimmt, bis alle möglichen Verschiebungs-Kombinationen berechnet

und die optimale Verschiebung bestimmt wurde. In der schematischen Darstellung in Abbildung 6 wäre das dritte Teilergebnis optimal, da alle Kantenpixel im Referenzkanal (grün) auf einem Kantenpixel im zu verschiebenden Bild (gelb) liegen.

Beispielsweise wird während des Kantenabgleich-Vorganges ein Pixel mit den Koordinaten (50,50) im Ursprungsbild bei einer `match_window_size` von 20 Pixel, einmal auf jedem Pixel von (40,40) bis (60,60) liegen.

Bestimmen Sie die optimalen Verschiebungen und transformieren Sie anschließend die Bilder des B-Kanals und G-Kanals indem Sie die Bilder entlang der x- und y-Achse gemäß der von Ihnen bestimmten optimalen Verschiebungen verschieben. Dabei ist es im Allgemeinen so, dass die berechneten Verschiebungen für die beiden Kanäle B und G nicht gleich sind. Speichern Sie Ihre Ergebnisse an die richtige Stelle des Vectors `aligned_images_BG`. Die Abmessungen der transformierten Bilder sollen mit jenen der Eingabebilder übereinstimmen. Abbildung 5 zeigt das Ergebnis des Kantenabgleichs zur Bestimmung der optimalen Translation.

### Nützliche Funktionen:

- Zugriff auf einzelnen Pixel in einem Grauwertbild: `Mat::at<uchar>(y, x)`

## 2.6 Kombinieren der Farbkanäle (2 Punkte)

Diese Aufgabe soll in der Funktion `combine_images(...)` implementiert werden. Nachdem Sie in den bisherigen Abschnitten die Kanäle so transformiert haben, dass sie nun bestmöglich übereinander liegen, sollen Sie in einem weiteren Schritt die Einzelbilder zu einem Gesamtbild mit drei Kanälen (RGB) kombinieren.

Bestimmen Sie also für jedes Pixel seinen (R,G,B)-Wert aus dem Referenzkanal bzw. den beiden im vorigen Schritt transformierten Kanälen. Achten Sie weiterhin auf die Korrektheit der Indizes – liegt ein Index außerhalb des gültigen Bereiches für einen Kanal,



Abbildung 6: Kantenabgleich (schematische Darstellung). Der dritte Schritt zeigt die optimale Translation.

so setzen Sie den Kanal des entsprechenden Pixels auf 0. Wiederholen Sie dies für alle weiteren Pixel und speichern Sie Ihr Ergebnis in der Matrix `output`. Für diese Übung soll die Matrix `output` dieselben Abmessungen wie der Referenzkanal (`image_R`) haben.

### Nützliche Funktionen:

- Zuweisung eines BGR-Pixels kann so durchgeführt werden:

```
Mat::at<cv::Vec3b>(y, x) = val
```

- Zugriff auf einzelnen Pixel in einem Grauwertbild: `Mat::at<uchar>(y, x)`

## 2.7 Zuschneiden des Resultats (2 Punkt)



(a) Kombinierte Bildkanäle.



(b) Ausgabe nach dem Zuschneiden.

Abbildung 7: Eingabe und Resultat des Zuschneide-Prozesses.

Diese Aufgabe soll in der Funktion `crop_image(...)` gelöst werden. Im vorangegangenen Schritt sind die Einzelkanäle wieder zu einem Farbbild kombiniert worden. Jedoch wurde nicht überprüft wo der Bild-Bereich endet in dem alle drei Kanäle gültige Werte haben. Daher kann es zu Rändern kommen, an denen ein oder mehrere Kanäle undefiniert sind, wie es in Abbildung 7 zu sehen ist. Um diese zu beseitigen, verkleinert man den Bildbereich. Dafür muss zuerst der Schnittbereich – also jener Bereich, der ausschließlich gültige Werte enthält – bestimmt werden. Er kann anhand der Abmessungen der Bilder – `img_r`, `img_g_aligned`, `img_b_aligned` – und der Verschiebungen zwischen den Kanälen R und G sowie, R und B bestimmt werden.

Abbildung 8 zeigt schematisch die einzelnen Kanäle (R, G und B), sowie den dazugehörigen Schnittbereich (schraffiert). Schneiden Sie diesen Bereich aus und speichern Sie ihn in out\_cropped.



Abbildung 8: Schnittbereich als Überlagerung der einzelnen Bild-Kanäle. Der schraffierte Bereich soll ausgeschnitten werden.

#### Nützliche Funktionen:

- `cv::max(...)`
- `cv::min(...)`
- `cv::Rect(...)`

### 3 Bonusaufgabe (3 Punkte)

Diese Aufgabe soll in der Funktion `bonus(...)` umgesetzt werden. Das Ziel der Bonusaufgabe ist es, einen Cartoon-Filter zu erstellen, der dann auf die vorhin erhaltenen Bilder angewendet wird. Das Ergebnis der Transformation kann in Bild 9 gesehen werden.

Dabei soll zunächst mit einem bilateralen Filter [2] das Eingangsbild geglättet werden, um einen cartoonmäßigen Look zu erzielen. Diese Filterart hat den Vorteil, dass sie sowohl radiometrische als auch geometrische Informationen im Bild berücksichtigt, um eine kantenerhaltende Glättung durchzuführen. Die Pixel des Output-Bildes werden folgendermaßen errechnet:

$$O(\mathbf{x}) = \frac{1}{W_p} \sum_{\mathbf{x}_i \in \omega} I(x_i) \cdot f_r(\|I(\mathbf{x}_i) - I(\mathbf{x})\|) \cdot g_s(\|\mathbf{x}_i - \mathbf{x}\|). \quad (9)$$



(a) Ergebnis der vorherigen Tasks

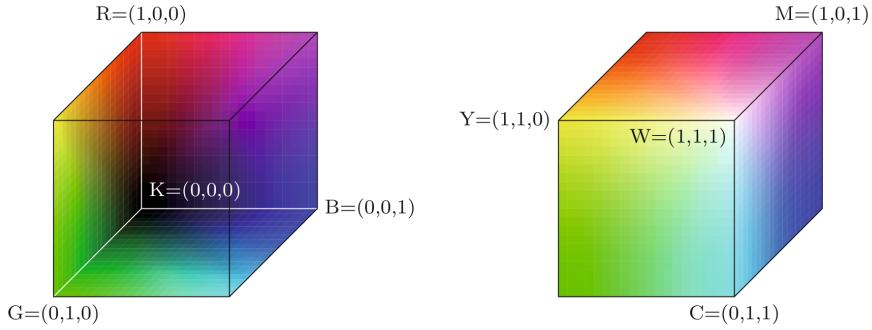
(b) Cartoon-Filter

Abbildung 9: Anwendungsbeispiel des Cartoon-Filters.

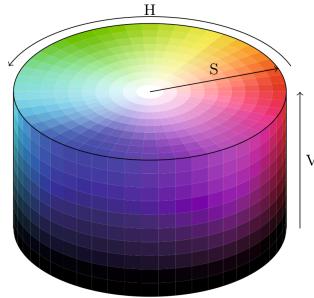
Dabei handelt es sich bei  $W_p$  um einen Normierungsfaktor,  $\mathbf{x} = (x, y)$  ist die Location des aktuell betrachteten Pixels und  $\mathbf{x}_i$  gibt die Position eines Pixels in der lokalen Nachbarschaft  $\omega$  an. In der Formel ist zu sehen, dass mit  $f_r$  zum einen die Intensitätswerte der Pixel und zum anderen mit  $g_s$  die räumliche Lage der Pixel zueinander verglichen werden. Dabei gilt sowohl für  $f_r$  als auch  $g_s$ , dass kleine Werte zurückgegeben werden, wenn große Farb- bzw. Lage-Unterschiede vorliegen. So wird erreicht, dass in uniformen Regionen eine gute Glättung auftritt, aber Kanten dennoch erhalten bleiben. In unserem Beispiel werden für  $f_r$  und  $g_s$  Gaußfilter verwendet. Diese Funktionalität kann mit dem Aufruf von `cv::bilateralFilter(...)` angewandt werden. Hier müssen für  $f_r$  und  $g_s$  die Sigma-Werte  $\sigma_r = 50$  und  $\sigma_s = 50$  übergeben werden. Außerdem soll für den Kernel-Durchmesser  $d = 7$  verwendet werden.

Nach diesem Schritt soll das Bild vom RGB-Farbraum mittels `cv::convertTo(...)` in den HSV-Raum transformiert werden, um die vorhandenen Farbwerte dort unter Zuhilfenahme einer bereitgestellten Lookup-Tabelle (LUT) zu quantisieren. In diesem Farbraum setzt sich die Beschreibung einer Pixelfarbe als *Hue*, *Saturation* und *Value* zusammen. Dabei wird mit H der Farbwert, mit S die Sättigung und mit V die Helligkeit angegeben. Bild 10 veranschaulicht die beiden Farbräume und gibt Aufschluss über den Zusammenhang zwischen RGB und HSV.

Hiermit soll ausdrücklich festgehalten werden, dass in der LUT Werte gespeichert sind, die im HSV-Farbraum liegen. Der Vorteil von dieser ist, dass aufwändige Berechnungen nur einmal im Vorfeld durchgeführt werden müssen. Anschließend kann in konstanter Zeit auf die erhaltenen Ergebnisse zugegriffen werden. Man erkaufte sich die Ausführungs-



(a) Der RGB-Farbraum visualisiert als Würfel. Die Intensitätswerte sind in dieser Abbildung auf das Intervall  $[0, 1]$  normiert. Jeder Farbwert kann entsprechend seiner R-, G- und B-Anteile eindeutig als Punkt in diesen Raum gemapped werden.



(b) Der HSV-Farbraum dargestellt als Zylinder. Man sieht, dass der H-Wert ein Winkel  $\in [0, 2\pi]$  ist, wo hingegen die Parameter S und V als Prozentwerte aufgefasst werden.

Abbildung 10: Gegenüberstellung der beiden Farbräume RGB (a) und HSV (b). Die Intervalle in denen diese Farbräume innerhalb von OpenCV abgebildet werden, können sich je nach Datentyp unterscheiden. Die Verwandtschaft zwischen RGB und HSV ist besonders deutlich sichtbar, wenn man sich den HSV-Zylinder als Diagonale im RGB-Cube vorstellt, die von Schwarz (K) nach Weiß (W) verläuft. Abbildung aus [1].

geschwindigkeit aber mit erhöhtem Speicherbedarf. Ein Pixel im HSV-Farbraum erhält mit der LUT folgendermaßen einen neuen Wert:

$$\text{pixel.val} = \text{lut}[\text{pixel.val}]. \quad (10)$$

Diese Zuweisung muss für jeden Farbkanal durchgeführt werden.

Nach diesem Schritt sollen noch die Kanten im Bild herausgearbeitet werden, um den charakteristischen Cartoon-Look zu verstärken. Dabei soll auf die schon implementierten Funktionen `compute_gradient(...)` und `compute_binary(...)` zurückgegriffen werden und auf das Graustufen-Bild angewendet werden. Mit diesen Funktionen und dem übergebenen Threshold `edge_threshold` soll wieder bestimmt werden, welche Pixel zu einer Kante gehören und welche nicht. Danach soll das Kanten-Bild mit einem Gaußfilter mittels

`cv::GaussianBlur(...)` geblurred werden, um einen feineren Übergang zu erhalten. Der Kernel soll hierbei die Größe 3x3 haben und  $\sigma_x = 0.6$  gesetzt werden.

Nun soll das quantisierte HSV-Bild mit den Kanten kombiniert werden. Hierzu muss der Value-Parameter des HSV-Bildes ja nach Wert des Kanten-Pixels gewichtet werden. Je stärker/heller die Kante ist, desto dunkler soll das Ausgangsbild und desto niedriger soll der Value-Parameter an dieser Stelle werden. Die Gewichtung soll daher wie folgt durchgeführt werden:

$$\text{value\_new} = \text{value\_old} * (1.0 - \text{edge\_val} / 255.0). \quad (11)$$

Zuletzt soll das resultierende kombinierte HSV-Bild wieder in den RGB-Farbraum rückkonvertiert werden. Das Ergebnis des Cartoon-Filters wird in Abbildung 9 gezeigt.

### Nützliche Funktionen:

- `cv::bilateralFilter(...)`
- `cv::cvtColor(...)`
- `cv::GaussianBlur(...)`

## 4 Ein- und Ausgabeparameter

Folgende Parameter sind in den Konfigurationsdateien angegeben:

- Dateiname
- Kanten Threshold
- Matching Window Size
- Dateinamen der Ausgabedateien
- Offset für die Translation

## 5 Programmgerüst

Die folgende Funktionalität ist in dem vom ICG zur Verfügung gestellten Programmgerüst bereits implementiert und muss von Ihnen nicht selbst programmiert werden:

- Die Konfigurationsdatei (JSON) wird vom Programmgerüst gelesen.
- Lesen des Eingabebildes und der Eingabeparameter
- Iteratives Ausführen der einzelnen Funktionen
- Schreiben der Ausgabebilder in die dafür vorgesehenen Ordner

## 6 Abgabe

Die Aufgaben bestehen jeweils aus mehreren Schritten, die zum Teil aufeinander aufbauen, jedoch unabhängig voneinander beurteilt werden. Dadurch ist einerseits eine objektive Beurteilung sichergestellt und andererseits gewährleistet, dass auch bei unvollständiger Lösung der Aufgaben Punkte erzielt werden können.

Wir weisen ausdrücklich darauf hin, dass die Übungsaufgaben von jedem Teilnehmer *eigenständig* gelöst werden müssen. Wenn Quellcode anderen Teilnehmern zugänglich gemacht wird (bewusst oder durch Vernachlässigung eines gewissen Mindestmaßes an Datensicherheit), wird das betreffende Beispiel bei allen Beteiligten mit 0 Punkten bewertet,

unabhängig davon, wer den Code ursprünglich erstellt hat. Ebenso ist es nicht zulässig, Code aus dem Internet, aus Büchern oder aus anderen Quellen zu verwenden. Es erfolgt sowohl eine automatische als auch eine manuelle Überprüfung auf Plagiate.

Die Abgabe der Übungsbeispiele und die Termineinteilung für die Abgabegespräche erfolgt über ein Webportal. Die Abgabe erfolgt ausschließlich über das Abgabesystem. Eine Abgabe auf andere Art und Weise (z.B. per Email) wird nicht akzeptiert. Der genaue Abgabeprozess ist im TeachCenter beschrieben.

Die Tests werden automatisch ausgeführt. Das Testsystem ist zusätzlich mit einem Timeout von 7 Minuten versehen. Sollte Ihr Programm innerhalb dieser Zeit nicht beendet werden, wird es vom Testsystem abgebrochen. Überprüfen Sie deshalb bei Ihrer Abgabe unbedingt die Laufzeit Ihres Programms.

Da die abgegebenen Programme halbautomatisch getestet werden, muss die Übergabe der Parameter mit Hilfe von entsprechenden Konfigurationsdateien genauso erfolgen wie bei den einzelnen Beispielen spezifiziert. Insbesondere ist eine interaktive Eingabe von Parametern nicht zulässig. Sollte aufgrund von Änderungen am Konfigurationssystem die Ausführung der abgegebenen Dateien mit den Testdaten fehlschlagen, wird das Beispiel mit 0 Punkten bewertet. Die Konfigurationsdateien liegen im JSON-Format vor, zu deren Auswertung steht Ihnen `rapidjson` zur Verfügung. Die Verwendung ist aus dem Programmgerüst ersichtlich.

Jede Konfigurationsdatei enthält zumindest einen Testfall und dessen Konfiguration. Es ist auch möglich, dass eine Konfigurationsdatei mehrere Testfälle enthält, um gemeinsame Parameter nicht mehrfach in verschiedenen Dateien spezifizieren zu müssen. In manchen Konfigurationsdateien finden sich auch einstellbare Parameter, die in Form eines `select` Feldes vorliegen. Diese sollen die Handhabung der Konfigurationsdateien erleichtern und ein einfaches Umschalten der Modi gewährleisten.

Es steht Ihnen frei, z.B. zu Testzwecken eigene Erweiterungen zu implementieren. Stellen Sie jedoch sicher, dass solche Erweiterungen in Ihrem abgegebenen Code deaktiviert sind, damit ein Vergleich der abgegebenen Arbeiten mit unserer Referenzimplementierung möglich ist.

Die Programmgerüste, die zur Verfügung gestellt werden, sind unmittelbar aus unserer Referenzimplementierung abgeleitet, indem nur jene Teile entfernt wurden, die dem Inhalt der Übung entsprechen. Die Verwendung dieser Gerüste ist nicht zwingend, aber Sie ersparen sich sehr viel Arbeit, wenn Sie davon Gebrauch machen.

## Literatur

- [1] Kristian Bredies and Dirk Lorenz. *Mathematische Bildverarbeitung*. Vieweg+Teubner Verlag, first edition, 2011.
- [2] Carlo Tomasi and Roberto Manduchi. Bilateral Filtering for Gray and Color Images. In *Proceedings of the IEEE International Conference on Computer Vision*, 1998.