CV/task1a — Vintage Effekt

Im Zuge der ersten Aufgabenstellung werden einige grundlegende Methoden und Vorgehensweisen der OpenCV-Library vorgestellt. Ziel der Übung ist es, einen *Vintage Color* Effekt auf ein Eingabebild anzuwenden, wobei der Effekt nur auf einen Teilbereich des Bildes anzuwenden ist (Abbildung 1b). Um dies zu erreichen, muss der Vordergrund vom Hintergrund mittels Schwellwert segmentiert und die resultierenden Bilder entsprechend kombiniert werden.

(a) (b)

Abbildung 1: Das originale Bild 1a und das endgültige Bild 1b in dem der *Vintage Color* Effekt angewandt wurde. Ausgenommen davon ist nur der Schriftzug der als Vordergrund segementiert wurde.

1 Aufgaben

Die Übung ist in *5* unterschiedliche Aufgabenstellungen unterteilt, die jeweils in einer eigenen Funktion im Framework zu implementieren sind. Alle **nicht** mit `TODO` gekennzeichneten Funktionen im Framework sollten **nicht** verändert werden. Die zu implementierenden Funktionen werden in folgende Aufgabenbereiche unterteilt:

- Anwendung des Vintage Color Effekts auf das gesamte Eingabebild
- Erstellen einer Maske für den Vordergrund durch Schwellwerte in einem vorgegebenen Bereich
- Verfeinerung der Maske durch closing
- Hinzufügen von Rauschen (noise) zu dem Bild
- Kombinieren des gefilterten Bildes mit dem Eingabebild unter Anwendung der zuvor erstellten Maske

Hinweis: Die für die OpenCV-Funktionen benötigten Parameter werden entweder in der Aufgabenstellung explizit erwähnt oder vom Framework bereitgestellt. Alle nicht in diesem Sinn erwähnten Parameter sind auf die Standard-default Werte zu setzen.

1.1 Anwendung des Vintage Color Effekts auf das gesamte Eingabebild (1 Punkte)

Dieser Teil ist in der Funktion *applyVintageColors(...)* zu implementieren. Ziel der Funktion ist es eine Punkttransformation auf das Eingabebild I anzuwenden. Die Punkttransformation ist im array *color_levels* definiert. Hierbei werden die im Framework bereits bereitgestellten Farben im Parameter *color_levels* an die Funktion übergeben. Die Farbe jedes Pixels im Bild soll nun auf die Farbe im dementsprechenden Channel der *color_levels* gesetzt werden. Die Werte im Vintage Bild ergeben sich durch folgenden Zusammenhang:

$$image\_color\_c(x,y) = color\_levels\_c(I\_c(x,y)), \tag{1}$$

c steht hierbei für den roten, grünen und blauen Farbkanal des Bildes. Das Resultat kann in Abbildung 2 betrachtet werden.

Hilfreiche OpenCV-Befehle:

- `cv::Mat::at<cv::Vec3b>(...)`

Abbildung 2: Das Bild nach Anwendung des *Vintage Color* Effekts.

- `cv::Vec3b`

1.2 Erstellung einer Maske für den Vordergrund (0.5 Punkte)

Dieser Teil ist in der Funktion *createMask(...)* zu implementieren. Die Vordergrund Segmentierung soll hierbei durch ein einfaches Schwellwerten des HSV-Farbraumes implementiert werden. In diesem Farbraum setzen sich Farben aus drei Kanälen zusammen:

- Farbwert (Hue)
- Sättigung (Saturation)
- Helligkeit (Value)

Die notwendigen unteren und oberen Schwellwertgrenzen werden für jedes Bild bereits vom Framework als Parameter an die Funktion übergeben (*lower_bound* und *upper_bound*). Zuerst sollte das Bild in den HSV Farbraum konvertiert werden. Danach sollte in dem durch den Parameter *location* festgelegten Bereich ein Thresholding mit den vorhin erwähnten unteren und oberen Grenzen stattfinden. Das daraus resultierende Maskenbild

ist in Abbildung 3 zu sehen. Als letzter Schritt soll das Maskenbild noch normalisiert werden, damit die Werte zwischen 0 und 1 liegen und dadurch die Binärmaske entsteht.

Abbildung 3: Das Maskenbild nach dem Thresholding

Hilfreiche OpenCV-Befehle:

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

1.2.1 Verfeinerung der Maske (3 Punkte)

Hinweis: Die folgenden Funktionen sind selbst zu implementieren, eine Verwendung diverser von OpenCV zur Verfügung gestellten Funktionen wie z.B. dilate(src, dilation_dst, structure_element) oder erode(src, erosion_dst, structure_element) ist untersagt.

Da eine reine Schwellwert-Segmentierung über Farbwerte nicht immer optimale Ergebnisse erzielt, wird in diesem Schritt die Maske noch verfeinert. Dieser Teil soll in den Funktionen *dilation(...)*, *erosion(...)* und *closing(...)* implementiert werden.
Diese morphologischen Basisoperationen arbeiten auf bereits segmentierten Bildern, wie es in der oben genannten Sektion 1.2 implementiert wurde. Man kann also annehmen, dass alle Pixel von Hintergrundsegmenten durch eine *0* und somit alle Vordergrundsegmente durch eine *1* gekennzeichnet sind. Ein ähnliches Verhalten zeigt sich auch bei den verwendeten Strukturelementen. Die gültigen Bereiche der Nachbarschaft eines Kernels weisen den Wert *1* auf, wonach die ungültigen Bereiche den Wert *0* besitzen.
Bei der Verwendung diese morphologischen Operationen muss darauf geachtet werden, dass die Vollständigkeit der ursprüngliche Form des Bildes so gut wie möglich erhalten bleibt und unerwünschte Objekte vollständig aus dem Bild verschwinden. Mit einer alleinigen Anwendung der Operationen Dilation oder Erosion kann dies nicht erreicht werden, jedoch gib es weitere morphologische Operationen wie closing oder opening, die zum gewünschten Effekt führen.

Dilation

Diese morphologische Operation beinhaltet die Faltung eines Bildes A mit einem Kernel X, der verschiedene Formen aufweisen kann. Normalerweise wird hierzu ein Quadrat oder ein Kreis verwendet. Der Kernel besitzt einen Ankerpunkt, welcher sich üblicherweise in der Mitte des Kernels befindet. Die Auswirkungen unterschiedlicher Ankerpunkte ist in Abbildung 4 dargestellt.

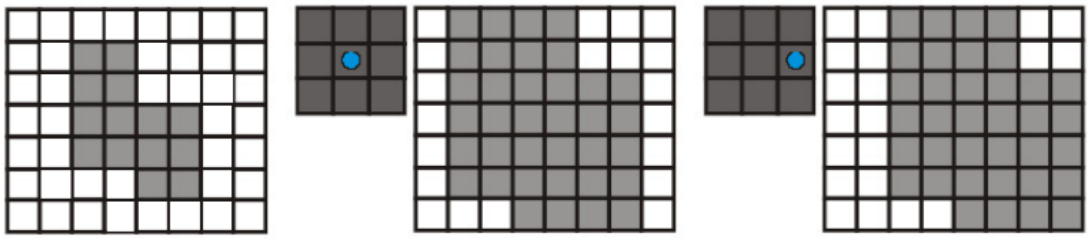

Abbildung 4: Dilation mit quadratischem Kernel und mittigem bzw. exzentrischem Ankerpunkt

Besitzt zumindest ein Pixel des Eingabebildes in der Nachbarschaft des Kernels den Wert *1*, so ist der Pixel des Bildes an der Position des Ankerpunktes auf *1* zu setzen. Die Nachbarschaftsregion ist durch den Parameter *kernel* gegeben. Ein Dilation-Schritt ist in Abbildung 5 dargestellt. Für unser Beispiel ist stets ein Ankerpunkt in der Mitte des Kernels anzunehmen. Desweiteren ist die Kernel-Größe stets eine ungerade Zahl $(3, 5, \ldots)$.
Hinweis zur Randbedingung: Der Kernel soll so über das Bild geschoben werden, dass der gesamte Kernel stets in das Bild passt. Die Ausgabepixel die zu nahe am Rand liegen und auf denen der Anker des Kernels nicht angewandt werden kann, sollen mit 0 befüllt werden.

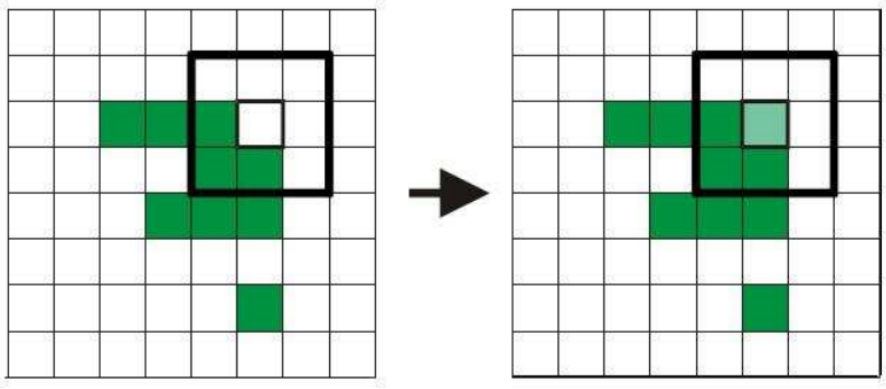

Abbildung 5: Dilation-Schritt an einem Pixel mit einem qadratischen Kernel

Ein passendes Video zu diesem Thema ist unter diesem Link[1] verfügbar.

Erosion

Diese Operation verhält sich ähnlich zur Dilation, jedoch wird hier ein lokales Minimum über die Kernelfläche berechnet. Besitzen alle Pixel des Eingabebildes in der Nachbarschaftsregion den Wert 1, so ist der Pixel des Bildes an der Position des Ankerpunktes auf 1 zu setzen. Die Nachbarschaftsregion ist durch den Parameter *kernel* gegeben. Ein Erosionsschritt ist in Abbildung 6 dargestellt.

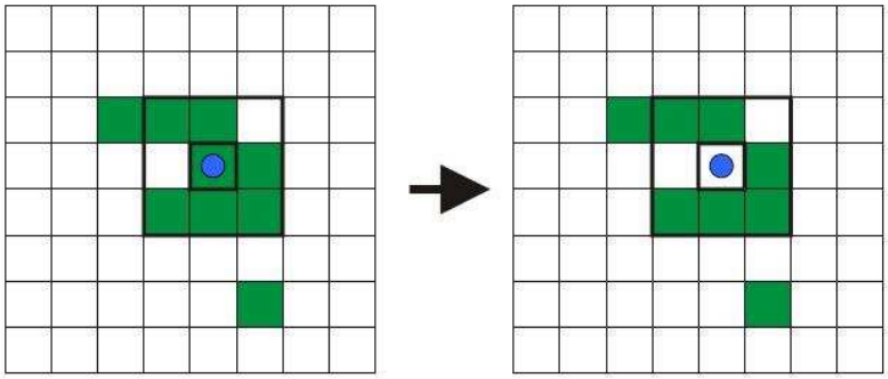

Abbildung 6: Erosion-Schritt an einem Pixel mit einem qadratischen Kernel

Für die Erosion ist, wie bei der Dilation, anzunehmen dass der Ankerpunkt in der Mitte des Kernels liegt und die Kernel-Größe stets eine ungerade Zahl ist. *Hinweis zur Randbedingung:* Der Kernel soll so über das Bild geschoben werden, dass der gesamte Kernel stets

[1] `https://www.youtube.com/watch?v=9lqH5XLI-V4`

in das Bild passt. Die Ausgabepixel die zu nahe am Rand liegen und auf denen der Anker des Kernels nicht angewandt werden kann, sollen mit 0 befüllt werden. Ein passendes Video zu diesem Thema ist unter diesem Link[2] verfügbar.

Closing

Das Closing ist eine weitere morphologische Operation, die sich aus einer Dilation, gefolgt von einer Erosion jeweils mit demselben Kernel zusammensetzt. Diese Operation kann auch wie folgt beschrieben werden:

$$(A \bullet X) = (A \oplus X) \ominus X, \tag{2}$$

wobei $\oplus$ eine Dilation bezeichnet und $\ominus$ eine Erosion bezeichnet. Zuerst werden durch die Dilation kleine Löcher innerhalb von Segmenten geschlossen, woraus sich auch die Bezeichnung *closing* ableitet. Die weiteren Effekte der Dilation wie eine Vergrößerung der Segmente werden durch die anschließende Erosion rückgängig gemacht. Diese morphologische Operation dient somit zum Überbrücken bzw. Schließen kleinerer Lücken und Löcher und zum Glätten innerer Ecken, wie in Abbildung 7 dargestellt.

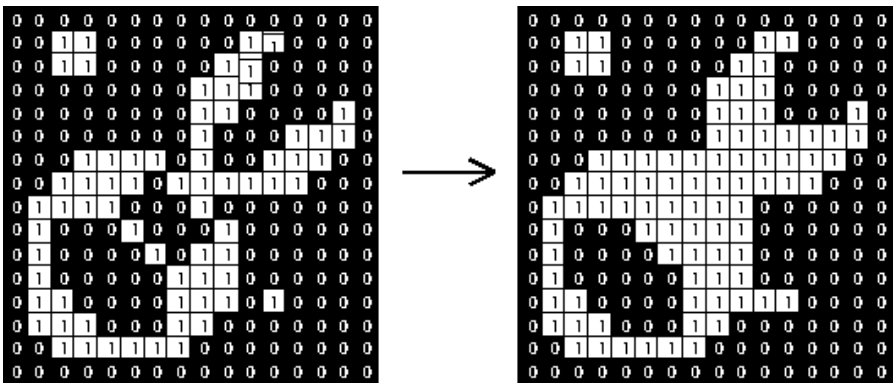

Abbildung 7: Closing-Operation an einem Beispiel

Hilfreiche OpenCV-Befehle:

- `cv::Mat::at<uint8_t>(...)`
- `cv::Mat::zeros(..)`
- `cv::Mat::copyTo(..)`

[2] `https://www.youtube.com/watch?v=rP1KZb3llCY`

1.3 Noise image (0.5 Punkte)

Um dem Vintagebild noch einen gewissen Feinschliff zu verleihen wird dem Bild ein leichtes Rauschen hinzugefügt. Diese Teilaufgabe ist in der Funktion *addNoise(...)* zu implementieren. Das Vintagebild wird mit Hilfe des als Parameter übergebenen Noisebildes kombiniert, wobei hier - ähnlich wie unter Sektion 1.1 erwähnt - zur Farbe jedes Pixels im Orignialbild der Wert von dem entsprechenden Channel des Noisebildes hinzu addiert wird. Die Werte ergeben sich durch folgenden Zusammenhang:

$$image\_result\_c(x,y) = color\_vintage\_c(x,y) + color\_noise\_c(x,y), \tag{3}$$

wobei c den Farbkanal bezeichnet. *Hinweis: Achten Sie darauf, dass Overflows (Werte > 255) korrekt gehandhabt (auf 255 abgeschnitten) werden.*

Hilfreiche OpenCV-Befehle:

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

1.4 Kombinieren des gefilterten Bildes mit dem Eingabebild (1 Punkte)

Dieser Teil ist in der Funktion *combineImages(...)* zu implementieren. Hier sollte das Originalbild mit dem Vintagebild mithilfe des Maskenbildes kombiniert werden. Das endgültige Bild sollte Pixel aus dem originalen, unbehandelten Bild enthalten wenn der Wert im Maskenbild an dieser Stelle *1* ist. Sollte der Wert an dieser Stelle *0* sein soll der Wert aus dem Vintagebild genommen werden. Ein Pixel des kombinierten Bildes K ergibt sich also aus dem Originalbild O, dem Vintage Bild V und der Maske nach Dilation und Erosion M nach folgendem Zusammenhang:

$$K(x,y) = \begin{cases} O(x,y), & \text{wenn } M(x,y) > 0 \\ V(x,y), & \text{sonst.} \end{cases} \tag{4}$$

Das kombinierte Bild ist in Abbildung 1b veranschaulicht.

2 Ein- und Ausgabeparameter

Die folgenden Parameter in der Konfigurationsdatei bestimmen das Verhalten des Programms (Anwendungsbeispiele finden Sie in den Dateien «`star_wars.json`», «`solo.json`»):

- Pfad des Eingabebildes: `input_path_img0`

- Pfad des Noise images: `input_path_noise`
- Pfad der Ausgabedateien: `output_path`
- Größe des Filter-Kerns: `kernel_size`
- Unterer und oberer Hue-Schwellwert zur Segmentierung des Vordergrunds `mask_hue_lower`, `mask_hue_upper`
- Unterer und oberer Saturation-Schwellwert zur Segmentierung des Vordergrunds `mask_sat_lower`, `mask_sat_upper`
- Unterer und oberer Value-Schwellwert zur Segmentierung des Vordergrunds `mask_val_lower`, `mask_val_upper`
- Position des Bereichs für die Vordergrund-Segmentierung: `label_x`, `label_y`, `label_width`, `label_height`

3 Programmgerüst

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

- Die JSON-Konfigurationsdatei wird vom Programmgerüst gelesen.
- Lesen der Eingabebilder.
- Lesen der `kernel_size`.
- Lesen des noise Bildes `noise_image`.
- Lesen der Schwellenwerte `mask_hue_lower`, `mask_hue_upper`, `mask_sat_lower`, `mask_sat_upper`, `mask_val_lower`, `mask_sat_upper`.
- Lesen von `label_x`, `label_y`, `label_width`, `label_height`.
- Schreiben der Ausgabebilder.
- Ihre Implementierungen sind nur in `main.cpp` durchzuführen.

Diese Aufgabe ist mit Hilfe von OpenCV[3] 2.4.9 zu implementieren. Nutzen Sie die Funktionen, die Ihnen OpenCV zur Verfügung stellt und achten Sie auf die unterschiedlichen Parameter und Bildtypen! Beachten Sie auch, dass die Anordung der Farbkanäle in OpenCV $[B,G,R]$ und nicht $[R,G,B]$ ist. Das C++ Cheatsheet[4] könnte bei dieser Aufgabe sehr hilfreich sein.

[3] http://opencv.org
[4] http://docs.opencv.org/2.4/opencv_cheatsheet.pdf

4 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.