

INSTITUT FÜR
INFORMATIONSVERARBEITUNG
LEIBNIZ UNIVERSITÄT HANNOVER

Appelstraße 9A
30167 Hannover

Labor zur Vorlesung
Digitale Bildverarbeitung

Datum: 26.06.2025
Uhrzeit: 17:00
Anzahl der Blätter: 21 (einschließlich Deckblatt)

Name:
YichenZhong
ChenlinLang
TranQuangThanh

Matrikelnummer:
10054755
10056439
10071713

Die Bearbeitung der Aufgaben erfolgt selbstständig in Kleingruppen. Alle Gruppenmitglieder sollen Arbeitsaufwand in gleicher Größenordnung einbringen. Betrugsversuche werden geahndet.

Inhaltsverzeichnis

1 Einleitung	3
--------------	---

1.1	Vorbereitung	4
1.2	Programmierumgebung	5
2	Aufgabe	8
2.1	Vorverarbeitung	10
2.1.1	Rauschreduktion	10
2.1.2	Histogramm Spreizung	11
2.2	Farbanalyse	13
2.2.1	RGB	13
2.2.2	HSV	15
2.3	Segmentierung und Bildmodifizierung	17
2.3.1	Statisches Schwellwertverfahren	17
2.3.2	Binärmaske	18
2.3.3	Bildmodifizierung	19
3	Zusammenfassung	21

1 Einleitung

Digitale Bildverarbeitung wird unter anderem in industriellen Prozessen, medizinischen Verfahren oder in Multimedia-Produkten angewandt. In diesem Labor soll eine beispielhafte Multimedia-Anwendung entwickelt werden, die über das Labor hinaus als erheiterndes Demonstrationsobjekt für Bildverarbeitung in Videokonferenzen genutzt werden kann. Als Inspiration dient ein Effekt aus der Filmreihe „Harry Potter“.



Abbildung 1: Harry Potter ohne magischen Tarnumhang

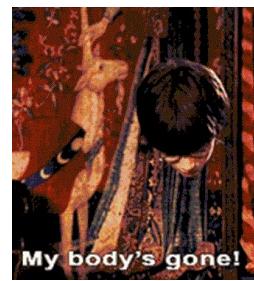


Abbildung 2: Harry Potter mit magischem Tarnumhang

Die Abbildungen 1 und 2 zeigen Harry Potter ohne¹ und mit² verzauberten Tarnumhang, welchen den Träger des Umhangs verschwinden lassen kann. In diesem Labor entwickeln Sie eine Bildverarbeitungs-Pipeline, mit welcher dieser Effekt mit einer einfachen Webcam simuliert werden kann. Sie festigen den Umgang mit

- Rauschunterdrückung
- Histogrammen
- verschiedenen Farbräumen
- Erosion und Dilatation
- Schwellwertverfahren

und üben gleichzeitig den Umgang mit der Programmiersprache Python.

¹<https://assets.cdn.moviepilot.de/files/8a2cc0d2eb31c41136fb2be242540606dcd50821980e830481404b2760fill/1280/614/Harry%20Potter%20Tarnumhang.jpg>

²<https://www.tres-click.com/wp-content/uploads/2019/06/harry-potter-tarnumhang.gif>

1.1 Vorbereitung

Für das Labor wird ein Rechner mit Betriebssystem Windows, Linux-Ubuntu oder Mac benötigt. Zusätzlich muss eine Webcam vorhanden sein (im Laptop integriert oder extern). Melden Sie sich **vorab** bei dem Betreuer des Labors, sollte Ihnen eines der benötigten Mittel nicht zur Verfügung stehen.

Um an dem Labor erfolgreich teilnehmen zu können, muss die zu nutzende Programmierumgebung vorbereitet werden. Sollten Sie an der praktischen Übung zur Vorlesung *Digitale Bildverarbeitung* teilgenommen haben, ist die erforderliche Programmierumgebung sowie der Programmcode wahrscheinlich bereits installiert und eingerichtet. Ansonsten befolgen Sie die Installationsanweisungen auf <https://github.com/TimoK93/Digitale-Bildverarbeitung> im Bereich *0_Einführung*. Die Installation des Streaming Programms zur Erstellung einer virtuellen Kamera ist dabei optional und nicht verpflichtend.

Nachdem Sie die Installation vollendet haben, öffnen Sie die Programmierumgebung, sodass Ihr Bildschirm Ähnliches zu Abbildung 3 anzeigt.

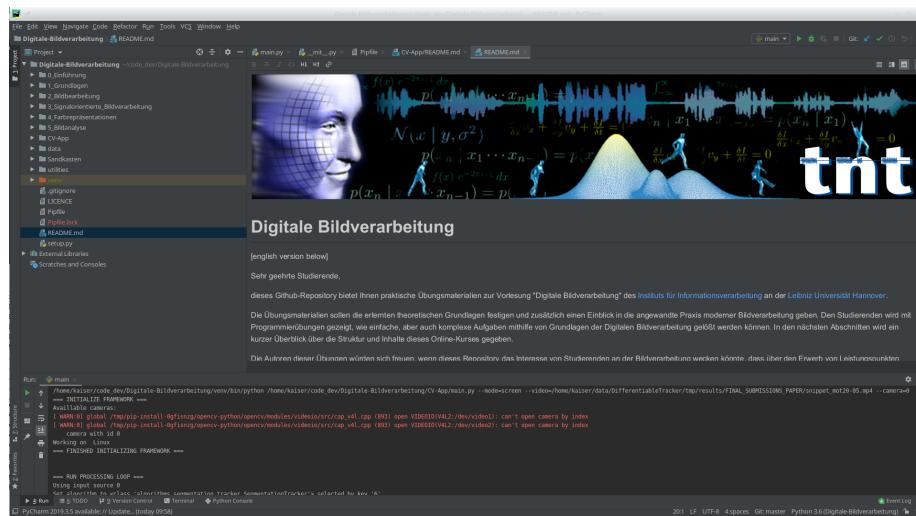


Abbildung 3: Programmierumgebung in PyCharm

Machen Sie sich als nächstes mithilfe der Beschreibung in Kapitel 1.2 mit der Programmierumgebung vertraut.

1.2 Programmierumgebung

In der Programmierumgebung ist das Hauptprogramm bereits soweit vorbereitet, so dass Sie einen Videostream aus Ihrer Kamera auslesen, darauf einen vorbereiteten Beispiyalgorithmus anwenden und diesen anzeigen können. In diesem Kapitel wird kurz erläutert, wie Sie diese Bildverarbeitungs-Pipeline starten und eigene Algorithmen integrieren können.

Starten der Bildverarbeitungs-Pipeline. Öffnen Sie die Datei `./CV-App/main.py` in PyCharm und klicken Sie mit der rechten Maustaste in den Programmcode. Öffnen Sie dann das Fenster *Modify Run Configuration...* wie in Abbildung 4 dargestellt. In dem Feld *Parameter* können Sie zusätzliche Argumente in die Umgebung eingeben. Argumente werden dabei im Schema `-Argument1 Value1 -Argument2 Value2` eingegeben. Die einzustellenden Parameter können Sie aus der Tabelle 1 entnehmen. Wenn Sie keine Argumente wählen, werden die Standard Einstellungen gewählt.

Tabelle 1: Argumente für die Programmausführung

Argument	Werte	Default-Wert
<i>camera</i>	-1, 0, 1, ... (-1 öffnet ein Video)	0
<i>mode</i>	<i>virtual_cam</i> (virtuelle Kamera), <i>screen</i> (nur Fenster)	<i>virtual_cam</i>
<i>video</i>	Pfad zu Video, wenn <i>camera</i> den Wert -1 hat.	-

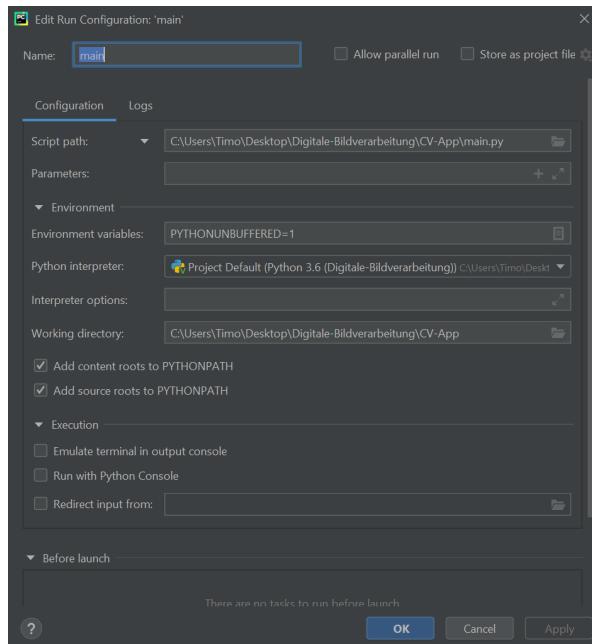


Abbildung 4: Run Configuration in PyCharm

Sie können das Programm nun mit *Run* starten. Kurz nach dem Start sollten Sie ein Fenster mit dem Videostream aus Ihrer Kamera sehen. Durch das Betätigen verschiedener Tasten auf der Tastatur (z.B. Taste 1 oder 2) können Sie verschiedene Algorithmen aktivieren. Sie sehen das Ergebnis direkt in dem ausgegebenem Videostream. Wichtig: Das Programm reagiert nur auf Tastendrücke, wenn das Videostream-Fenster im Vordergrund ihres Bildschirms ist.

Eigene Algorithmen einbinden. Für Sie sind nur die Dateien im Dateipfad */CV-App/algorithms* relevant. Beispielhaft erstellen Sie im Folgenden den Algorithmus *YourAlgorithm*.

Erstellen Sie eine Datei *your_algorithm.py* und kopieren Sie den Inhalt aus dem Code 1. In der Funktion *__init__(self)* können Sie dauerhafte Variablen definieren. Die Funktion *process(self, img)* verarbeitet das Eingangsbild *img* und gibt es am Ende modifiziert wieder aus (Hinweis: Ein und Ausgangsbild müssen gleich groß sein!). Die Funktion *mouse_callback(self, ...)* reagiert auf Aktionen der Maus, wie zum Beispiel einem Mausklick an der Position x und y. So können Sie mit ihrem Algorithmus interagieren. Sie können sich einige Beispielalgorithmen in dem Ordner */CV-App/algorithms* zur Veranschaulichung ansehen. Der Algorithmus in *white_balancing.py* veranschaulicht alle Funktionen mit Algorithmen der Klasse *Algorithm*.

Code 1: Eigener Algorithmus in *your_algorithm.py*

```
1 import cv2
2 import numpy as np
3 from . import Algorithm
4
5 class YourAlgorithm(Algorithm):
6     def __init__(self):
7         self.some_persistent_value = "i_am_always_existing"
8
9     def process(self, img):
10        print(self.some_persistent_value)
11        return img
12
13    def mouse_callback(self, event, x, y, flags, param):
14        if event == cv2.EVENT_LBUTTONUP:
15            print("A_Mouse_Click_happend!_at_position", x, y)
```

Um Ihren Algorithmus nun mit einer Aktivierungstaste-Taste zu verlinken, öffnen Sie die Datei `__init__.py`. Am Ende der Datei ist eine Verlinkung der Algorithmen zu bestimmten Tasten zu sehen (siehe Code 8).. In dem Beispiel in Code 8 ist Ihr neuer Algorithmus `YourAlgorithm` importiert und an die Taste 3 verlinkt.

Code 2: Verlinkung der Algorithmen in `__init__.py`

```
1 from .image_to_gray import ImageToGray
2 from .image_to_hue import ImageToHue
3 from .your_algorithm import YourAlgorithm
4
5 algorithms = dict()
6 algorithms["0"] = Algorithm
7 algorithms["1"] = ImageToGray
8 algorithms["2"] = ImageToHue
9 algorithms["3"] = YourAlgorithm
```

Nach Neustart des Programms durch erneute Betätigung der *Run* Funktion, ist Ihr Algorithmus durch betätigen der Taste 3 zugänglich.

2 Aufgabe

Das Ziel ist die Detektion des „magischen Umhangs“ sowie das künstliche Entfernen eines Objektes im Vordergrund. Dafür werden in diesem Labor drei Arbeitspakete bearbeitet: Die Vorverarbeitung, die Farbanalyse der Szene und die Segmentierung des Umhangs. Eine Skizze der Bildverarbeitungs-Pipeline ist in Abbildung 5 dargestellt.

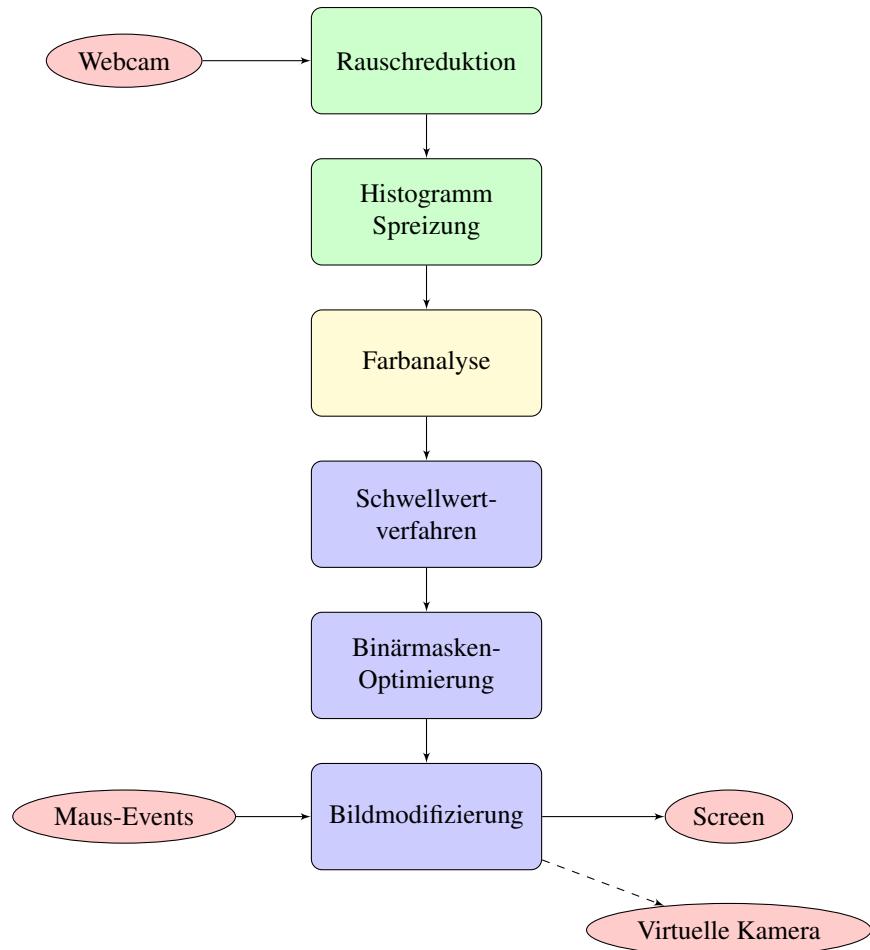


Abbildung 5: Bildverarbeitungs-Pipeline

Die Szene für diesen Versuch wird durch Sie definiert: Wählen Sie sich eine eintönige, möglichst monotone Umgebung als Szene für diesen Versuch. Der „magische Umhang“ wird dann durch einen einfarbigen Gegenstand (es muss kein Umhang sein!) realisiert. Achten Sie darauf, dass sich der Umhang farblich von der Szene unterscheidet. Je stärker der Kontrast zwischen „Umhang“ und Szene ist, desto besser lässt sich

dieser Versuch durchführen.

Im folgenden finden Sie detaillierte Beschreibungen der Arbeitspakete. Bitte beantworten Sie die vorhandenen Fragen und erstellen Sie ggf. geforderten Code oder Abbildungen. Die Bearbeitung der Fragen kann entweder innerhalb dieses Latex Dokuments, oder in einem separatem PDF geschehen.

2.1 Vorverarbeitung

Um die folgende Verarbeitung der Bilder zu vereinfachen und robuster zu gestalten, sollen Sie den Videostream mit einem Preprocessing vorverarbeiten. Binden Sie dafür die Datei `./CV-App/algorithms/invis_cloak.py` in den Algorithmus ein, wie in der Einleitung beschrieben. Die folgenden Aufgabenstellungen sind in den dafür vorgesehenen Funktionen zu bearbeiten.

2.1.1 Rauschreduktion

Jeder Farbwert eines Pixels $I_k(x, y) \in \{0, \dots, 255\}$ mit $k \in \{R, G, B\}$ wird auf dem Kamerasensor durch einen elektrischen Halbleiter physikalisch gemessen. Je nach Sensorqualität und Lichtbedingungen wirkt dabei ein unterschiedlich ausgeprägtes Rauschen auf die Farbwerte ein, sodass der zur Verfügung stehende Farbwert als Summe

$$I_k(x, y) = I_k^*(x, y) + r(x, y) \quad (1)$$

aus realem Farbwert $I_k^*(x, y)$ und statistischem Rauschen $r(x, y)$ modelliert werden kann. Das Rauschen r kann als normalverteilt um den Mittelwert 0 angenommen werden. Unter den Annahmen, dass die Kamera statisch montiert ist und in der aufgenommenen Szene keine Veränderung passiert, kann der Zusammenhang

$$\bar{I}_{k,t}(x, y) = \lim_{N \rightarrow \infty} \frac{1}{N+1} \sum_{n=0}^N I_{k,t-n}^*(x, y) + r_{t-n}(x, y) \stackrel{!}{=} I_{k,t}^* \quad (2)$$

für die Mittelwertbildung über lange Zeiträume formuliert werden. Dabei beschreibt t den Zeitpunkt, zu dem der entsprechende Wert gemessen wurde.

Um die Bildqualität zu erhöhen, soll der Einfluss von r reduziert werden. Es soll dafür angenommen werden, dass die Kamera statisch ist und kaum Bewegung in zwei aufeinander folgenden Bildern vorhanden ist. Implementieren Sie die Mittelwertbildung mit einer variablen Bildreihe N (default: $N = 1$) und geben Sie das Bild aus.

Um zu prüfen wie das Bild auf Pixelebene arbeitet, kann die Variable `plotNoise` in der Funktion `process()` auf `True` gesetzt werden. Es werden zwei zusätzliche Plots ausgegeben, in der ein Bildausschnitt des Zentrums vor- und nach der Rauschunterdrückung vergrößert dargestellt werden.

Aufgabe 1 Geben Sie Ihren Code an und beschreiben Sie ihn. Geben Sie nur relevante Code Bereiche an!

Code 3: Vorverarbeitung, Aufgabe 1

```
1 # Your code!
2 def _211_Rauschreduktion(self, img):
3     """
```

```

4      Hier steht Ihr Code zu Aufgabe 2.1.1 (Rauschunterdrückung)
5      - Implementierung Mittelwertbildung über N Frames
6
7      N = 5 # Number of frames to be averaged, can be set externally as a parameter if necessary
8
9      if not hasattr(self, "frame_buffer"):
10         self.frame_buffer = []
11
12     # Add the current frame to the buffer
13     self.frame_buffer.append(img.copy())
14
15     # If there are more than N frames, delete the oldest
16     if len(self.frame_buffer) > N:
17         self.frame_buffer.pop(0)
18
19     # Convert to float32 for averaging to prevent overflow.
20     img = np.mean(np.array(self.frame_buffer).astype(np.float32), axis=0)
21
22     # Turning back to the uint8 image
23     img = np.clip(img, 0, 255).astype(np.uint8)
24
25     return img

```

Aufgabe 2 Nennen Sie Vor und Nachteile, wenn N vergrößert werden würde. Sollte N in dieser Anwendung vergrößert werden?

Vorteil: Durchschnittswert von Rauschen wird verringert.

Nachteil: Das Verarbeitung ist zeitaufwändig. Deswegen ist Kamera langsamer

Aufgabe 3 Beschreiben Sie eine weitere Methode zur Rauschreduktion. Diskutieren Sie dabei Vor- oder Nachteile!

Box-Filter: ein Mittelwertfilter. Einfach, aber stumpf.

Binomialfilter: diskrete Annäherung an den Gauß-Filter. Einfach, aber bewahrt Details besser als der Box-Filter.

Gauß-Filter: einfach und effektiv. Medianfilter: ein nichtlinearer Filter. Robust beim Entfernen von Salz-und-Pfeffer-Rauschen.

2.1.2 Histogramm Spreizung

Pixel können in unserer Anwendung Werte von $I_k(x, y) \in \{0, \dots, 255\}$ annehmen. Dieser Wertebereich wird nicht zwangsläufig ausgenutzt. Um das zu ändern, soll eine Histogramm Spreizung auf den Helligkeitsinformationen der Pixel durchgeführt werden.

Implementieren Sie zusätzlich zur Rauschreduktion eine Histogramm Spreizung, indem sie (1) das Rausch-reduzierte Eingangsbild in den HSV-Farbbereich transformieren und (2) die Rechenvorschrift 3 auf den V-Kanal anwenden. Transformieren Sie das Bild dann (3) wieder in den RGB Farbraum.

$$I_V^{\text{new}}(x, y) = \frac{I_V(x, y) - \min I_V}{\max I_V - \min I_V} \cdot 255 \quad (3)$$

Hinweis: Nutzen Sie die Befehle `cv2.cvtColor(img, cv2.COLOR_BGR2HSV)` beziehungsweise `cv2.cvtColor(img, cv2.COLOR_HSV2BGR)`.

Aufgabe 4 Geben Sie Ihren Code an und beschreiben Sie ihn. Geben Sie nur relevante Code Bereiche an!

Code 4: Vorverarbeitung, Aufgabe 4

```

1 # Your code!
2 def _212_Histogrammspreizung(self, img):
3     """
4         Hier steht Ihr Code zu Aufgabe 2.1.2 (Histogrammspreizung)
5         - Transformation HSV
6         - Histogrammspreizung berechnen
7         - Transformation BGR
8     """
9     # Convert to HSV
10    hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
11    h, s, v = cv2.split(hsv)
12
13    # Calculate V-channel minimum and maximum values
14    v_min = np.min(v)
15    v_max = np.max(v)
16
17    # Avoid dividing by 0
18    if v_max - v_min == 0:
19        v_stretched = v
20    else:
21        v_stretched = ((v - v_min) / (v_max - v_min) * 255).astype(np.uint8)
22
23    # Merge and convert back to BGR
24    hsv_stretched = cv2.merge([h, s, v_stretched])
25    img = cv2.cvtColor(hsv_stretched, cv2.COLOR_HSV2BGR)
26
27    return img

```

Aufgabe 5 Warum ist es sinnvoll, den gesamten Wertebereich für die Darstellung von Videos in Multimedia-Anwendungen auszunutzen?

Weil durch die Histogrammspreizung der ganze Wertebereich (0–255) genutzt wird und so Kontrast und Details im Bild verbessert werden.

2.2 Farbanalyse

Die Zugrundeliegende Aufgabe ist die Detektion des „magischen Umhangs“, der Objekte verschwinden lassen kann. Der Umhang kann durch eine einfarbige Decke modelliert werden. Sollte keine einfarbige Decke vorhanden sein, kann ebenfalls ein einfarbiges Blatt Papier zur Hilfe genommen werden.

Das Ziel des Arbeitspakets „Farbanalyse“ ist es, die farblichen Eigenschaften der Szene, sowie des „magischen Umhangs“ zu untersuchen. Dafür sollen die Histogramme einzelner Farbkanäle mit und ohne Umhang erstellt und analysiert werden.

Hinweis: Versuchen Sie von nun an die Position der Kamera nicht mehr zu verändern!

2.2.1 RGB

Einzelne Pixel werden durch drei Werte repräsentiert. Jeweils ein Wert $I_k(x, y) \in \{0, \dots, 255\}$ mit $k \in \{R, G, B\}$ beschreibt die einfallende Lichtmenge für die Farben Rot, Grün und Blau. In OpenCV werden Bilder im BGR Format repräsentiert. Die Verteilung der Farbwerte kann durch ein Histogramm dargestellt werden. Ein Histogramm

$$h(v) = |I_v| \quad (4)$$

beschreibt die Anzahl der Menge Pixel I_v im Bild, welche den Wert v haben. In OpenCV kann das Histogramm mit der Funktion `cv2.calcHist(image, [Kanal], None, [histSize], histRange, False)` berechnet werden. Dabei gibt `histSize` die Anzahl der Intervalle und `histRange = (0, 256)` die obere und untere Schrank für den zu betrachtenden Wertebereich an.

Implementieren Sie in Ihren Algorithmus eine Funktion, mit dem Sie per Mausklick das aktuelle Bild speichern können. Des Weiteren soll bei Betätigung des Mausklicks ein Histogramm für jeden Farbkanal des RGB-Bilder erstellt und abgespeichert werden. Mit Hilfe des Code-Schnipsel in Code 5 kann ein Histogramm angezeigt oder gespeichert werden!

Code 5: Histogrammberechnung mit *matplotlib*

```
1 import cv2
2 from matplotlib import pyplot as plt
3
4 channel = 0 #[0:B, 1:G, 2:R]
5 hist_size = 256
6 hist_range = [0,256]
7 histr = cv2.calcHist([img], [channel], None, [hist_size], hist_range)
8 plt.plot(histr, color = "b")
9 plt.xlim([0,256])
10 plt.savefig('the_path_to_store.png')
11 plt.show()
```

Nehmen Sie mit dem fertig implementierten Code ein Bild und die Histogramme in der von Ihnen präferierten Szene auf. Nehmen Sie sich darauf den „magischen Um-

hang“ zur Seite und halten ihn sehr gut sichtbar vor die Kamera. Nehmen Sie auch jetzt ein Bild mit den Histogrammen auf. Die Kamera sollte sich zwischen den beiden Bildern nicht bewegen.

Aufgabe 1 Geben Sie Ihren Code an und beschreiben Sie ihn. Geben Sie nur relevante Code Bereiche an! Geben sie zusätzlich die aufgenommenen Bilder und die erstellten Histogramme an.

Code 6: Farbanalyse, Aufgabe 1

```

1 # Your code!
2 def _221_RGB(self, img):
3     """
4         Hier steht Ihr Code zu Aufgabe 2.2.1 (RGB)
5         - Histogrammberechnung und Analyse
6     """
7     if not self.capture_for_rgb_analysis:
8         return # No processing when there are no clicks
9
10    print("saving...")
11
12    # Save current image
13    cv2.imwrite("rgb_input_image.png", img)
14
15    # Plotting RGB three-channel histograms
16    colors = ('b', 'g', 'r')
17    plt.figure()
18    for i, col in enumerate(colors):
19        hist = cv2.calcHist([img], [i], None, [256], [0, 256])
20        plt.plot(hist, color=col, label=f"{col.upper()}-Kanal")
21        plt.xlim([0, 256])
22
23    plt.title("RGB_Histogramm")
24    plt.xlabel("Pixelwert")
25    plt.ylabel("Anzahl_der_Pixel")
26    plt.legend()
27    plt.grid(True)
28    plt.tight_layout()
29    plt.savefig("rgb_histogram.png")
30    plt.close()
31
32    print("saved:rgb_input_image.png_rgb_histogram.png")

```

Aufgabe 2 Interpretieren Sie die Veränderungen zwischen den Histogrammen mit und ohne „magischen Umhang“. Verhalten sich die einzelnen Kanäle gleich? Lassen sich Bereiche in den Histogrammen herausstellen, die dem Umhang zuzuordnen sind? Diskutieren Sie Ihre Beobachtungen.

Der Umhang zeigt sich deutlich im Histogramm, vor allem im Farbkanal, der seiner Farbe entspricht. Die Kanäle verhalten sich unterschiedlich – der betroffene Kanal hat

einen starken Ausschlag. So lassen sich die Farbwerte des Umhangs klar erkennen und gezielt herausfiltern.

2.2.2 HSV

Erweitern Sie ihren vorherigen Code um eine Farbkonvertierung in den HSV-Farbraum. Führen Sie die Konvertierung vor Erstellung der Histogramme durch und wiederholen Sie die Schritte aus dem vorherigen Aufgabenteil.

Aufgabe 3 Geben sie die aufgenommenen Bilder und die erstellten Histogramme an.



Abbildung 6: frame 1

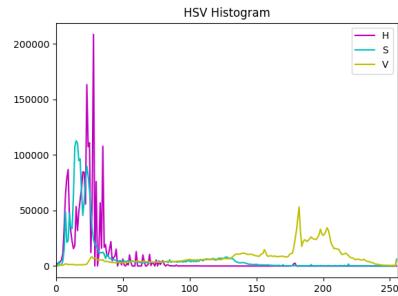


Abbildung 7: Histogramm 1



Abbildung 8: frame 2

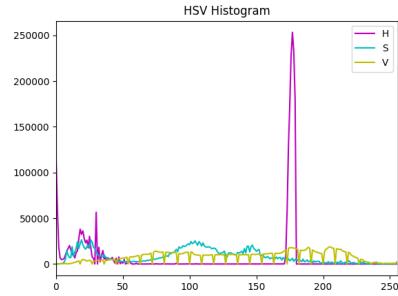


Abbildung 9: hist 2

Aufgabe 4 Interpretieren Sie die Veränderungen zwischen den Histogrammen mit und ohne „magischen Umhang“. Verhalten sich die einzelnen Kanäle gleich? Lassen sich Bereiche in den Histogrammen herausstellen, die dem Umhang zuzuordnen sind? Diskutieren Sie Ihre Beobachtungen.

Mit Umhang ändert sich das Histogramm deutlich, besonders im H-Kanal. Es entstehen neue Peaks, die dem Umhang zugeordnet werden können. Die Kanäle verhalten sich unterschiedlich – vor allem der Farbton zeigt starke Veränderungen.

Aufgabe 5 Versuchen Sie mit den gegebenen Histogrammen Wertebereiche zu finden, mit denen Sie den „magischen Umhang“ segmentieren könnten. Formulieren Sie eine Regel in dem Format

$$S_{\text{Umhang}} = \{I(x, y) \mid R_{\min} < I_R(x, y) < R_{\max} \text{ und } \dots\}, \quad (5)$$

wobei S_{Umhang} die Binärmaske beschreibt und R_{\min} und R_{\max} beispielhafte Schwellwerte für den Rot-Kanal sind.

Für den Umhang lassen sich im H-Kanal Werte z.B. zwischen 160 und 180 beobachten. Diese Schwelle trennt den Umhang gut vom Rest des Bildes.

Aufgabe 6 Worauf muss geachtet werden, wenn mit dem H-Kanal des HSV-Farbraums gearbeitet wird?

Rot liegt zum Beispiel bei H-Werten um 0 und 179. Außerdem verwendet OpenCV den Bereich 0 bis 179 statt 0 bis 360.

2.3 Segmentierung und Bildmodifizierung

In diesem Arbeitspaket werden Sie auf Grundlage der vorherigen Analysen eine Segmentierung des magischen Umhangs realisieren. Anschließend werden Sie den segmentierten Bereich „verschwinden“ lassen, indem sie ein statisches Bild des Hintergrunds auf diese Flächen einfügen.

2.3.1 Statisches Schwellwertverfahren

Implementieren Sie die von Ihnen gefundene Regel nach Gleichung 5, um eine Binärmaske zu erhalten. Sie können die Randbedingungen wie im folgenden Code-Schnipsel 7 implementieren.

Code 7: Benutzung von Randbedingungen mit *numpy*

```
1 import cv2
2 import numpy as np
3
4 channel1 = 0
5 lower_bound1, upper_bound1 = 15, 100
6 is_condition_1_true = (lower_bound1 < img[:, :, channel1]) * \
7     (img[:, :, channel1] < upper_bound1)
8 channel2 = 1
9 lower_bound2, upper_bound2 = 65, 172
10 is_condition_2_true = (lower_bound2 < img[:, :, channel2]) * \
11     (img[:, :, channel2] < upper_bound2)
12
13 binary_mask = is_condition_1_true * is_condition_2_true
```

Geben Sie die gefundene Binärmasken als Ausgangsbild auf dem Bildschirm aus. Sollten die gefundenen Wertebereich zu keinen sinnvollen Segmentierungen führen, dürfen Sie Gleichung 5 selbstverständlich anpassen!

Implementieren Sie ebenfalls eine Mausklick-Funktion, mit der Sie das aktuelle Bild und die dazugehörige Binärmaske abspeichern können. Für das Abspeichern von Bildern können Sie die Funktion *cv2.imwrite(img, "path_to_store.png")* verwenden.

Aufgabe 1 Geben Sie Ihren Code an und beschreiben Sie ihn. Geben Sie nur relevante Code Bereiche an! Geben Sie ebenfalls das aufgenommene Bild sowie die dazugehörige Binärmaske an.

Code 8: Segmentierung und Bildmodifizierung, Aufgabe 1

```
1 # Your code!
2 hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
3
4 # Define the red range
5 lower_red1 = np.array([0, 150, 120])
6 upper_red1 = np.array([10, 255, 255])
7 lower_red2 = np.array([170, 150, 120])
```

```

8     upper_red2 = np.array([180, 255, 255])
9
10    # Generate Mask
11    mask1 = cv2.inRange(hsv, lower_red1, upper_red1)
12    mask2 = cv2.inRange(hsv, lower_red2, upper_red2)
13    mask = cv2.bitwise_or(mask1, mask2)
14
15    self.binary_mask = mask

```

2.3.2 Binärmaske

Die in der vorherigen Aufgabe erhaltene Binärmaske ist ggf. ungeeignet für eine zufriedenstellende Segmentierung. Sie sollen die Maske nun optimieren. Wenden Sie dafür das *Opening* und *Closing* auf die Binärmaske an. Nutzen Sie die Funktionen `cv2.erode(img, kernel)` und `cv2.dilate(img, kernel)`.

Wählen Sie zum Schluss die größte zusammenhängende Region segmentierter Pixel aus, und löschen alle anderen Segmente. Folgender Code-Schnipsel 11 soll als Hilfestellung dienen. Recherchieren Sie ggf. im Internet.

Code 9: Konturfindung

```

1 (cnts, _) = cv2.findContours(
2         binary_mask, cv2.RETR_LIST, cv2.CHAIN_APPROX_SIMPLE)
3 c = max(cnts, key = cv2.contourArea)
4 img = cv2.drawContours(img, [c], -1, color=255, -1)

```

Aufgabe 2 Geben Sie Ihren Code an und beschreiben Sie ihn. Geben Sie nur relevante Code Bereiche an!

Code 10: Segmentierung und Bildmodifizierung, Aufgabe 2

```

1 # Your code!
2 kernel = np.ones((5, 5), np.uint8)
3
4 mask_clean = cv2.morphologyEx(self.binary_mask, cv2.MORPH_OPEN, kernel, iterations=2)
5 mask_clean = cv2.morphologyEx(mask_clean, cv2.MORPH_CLOSE, kernel, iterations=2)
6
7 # Find the maximum connectivity area
8 num_labels, labels, stats, centroids = cv2.connectedComponentsWithStats(mask_clean, connecti
9
10 if num_labels > 1:
11     max_label = 1 + np.argmax(stats[1:, cv2.CC_STAT_AREA])
12     mask_largest = np.uint8(labels == max_label) * 255
13 else:
14     mask_largest = mask_clean
15
16 self.binary_mask = mask_largest

```

Aufgabe 3 Welche Probleme oder Fehler können in der Binärmaske vorkommen, die mit den Maßnahmen beseitigt werden sollen?

In der Binärmaske können kleine Löcher, Rauschen oder unerwünschte Punkte auftreten. Diese Fehler sollen mit Morphologie-Operationen wie Öffnen oder Schließen entfernt werden.

2.3.3 Bildmodifizierung

Nach dem Fertigstellen der vorherigen Aufgabenstellungen sollten Sie nun eine Binärmaske erhalten, welche den „magischen Umhang“ segmentiert. Die letzte Aufgabe befasst sich mit der Bildmodifizierung, welche den Eindruck verschwindender Objekte vermittelt.

Sie sollen nun folgende Funktionen implementieren:

Erstellen Sie eine Member-Variable (z.B. *self.variable*) in die Algorithmus Funktion *_init_()*. Initieren Sie die Variable mit dem Wert *None*. Modifizieren Sie den Algorithmus, sodass Sie mit einem Mausklick ein Bild in die Variable speichern können. Dieses Bild wird als Hintergrund definiert. Mausklick-Funktionen aus vorherigen Aufgaben können überschrieben werden!

Solange kein Bild in der Variable gespeichert ist, soll das Eingangsbild direkt wieder ausgegeben werden. Sobald ein Hintergrund vorhanden ist soll folgendes passieren: Modifizieren Sie das Bild, indem Sie das Ausgangsbild aus dem derzeitigen Kamera-Stream und dem Hintergrund zusammenfügen. Die durch die Binärmaske segmentierte Fläche soll aus dem Hintergrund entnommen werden, die unsegmentierte Fläche aus dem derzeitigen Videostream.

Hinweis: Verlassen Sie das Sichtfeld der Kamera, während Sie die Hintergrund Aufnahme aufnehmen.

Aufgabe 4 Geben Sie Ihren Code an und beschreiben Sie ihn. Geben Sie nur relevante Code Bereiche an!

Code 11: Segmentierung und Bildmodifizierung, Aufgabe 4

```
1 # Your code!
2 if not hasattr(self, 'background'):
3     self.background = None
4     if not hasattr(self, 'click_triggered'):
5         self.click_triggered = False
6
7     if self.click_triggered and self.background is None:
8         self.background = img.copy()
9         print("Background_has_been_saved.")
10        return img
11
12    if self.background is None:
```

```
13         return img
14
15     # Create foreground mask
16     mask_inv = cv2.bitwise_not(self.binary_mask)
17
18     # Extract the current image portion of the non-cloaked area
19     fg = cv2.bitwise_and(img, img, mask=mask_inv)
20
21     # Extract the background portion of the cloak area
22     bg = cv2.bitwise_and(self.background, self.background, mask=self.binary_mask)
23
24     img = cv2.add(fg, bg)
```

Aufgabe 5 Geben Sie ein Bild (z.B. Screenshot) an, in dem die Funktion Ihres „magischen Umhangs“ gezeigt wird!

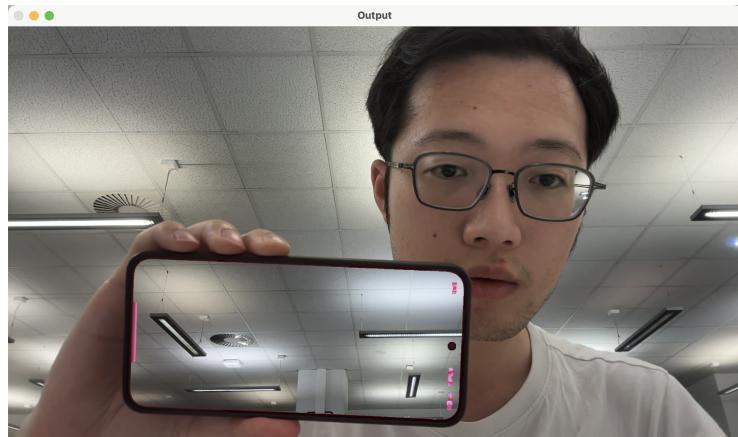


Abbildung 10: Rote Tapete als Umhang

3 Zusammenfassung

Herzlichen Glückwunsch! Sie haben in diesem Labor eine Bildverarbeitungs-Pipeline erfolgreich implementiert. Bitte fassen Sie die genutzten Methoden und Erkenntnisse kurz zusammen.

In diesem Labor haben wir eine Bildverarbeitungs-Pipeline aufgebaut. Zuerst wurde das Bildrauschen reduziert, indem mehrere Bilder gemittelt wurden. Danach haben wir eine Histogrammspreizung durchgeführt, um den Kontrast zu verbessern. Anschließend wurde das Bild in den HSV-Farbraum umgewandelt, um den Umhang anhand seiner Farbe zu erkennen. Wir haben Histogramme im RGB- und HSV-Raum erstellt, um die Farbverteilung zu analysieren. Dann wurde mit Hilfe einer Maske der Umhang im Bild markiert. Diese Maske wurde mit Morphologie verbessert, um Störungen zu entfernen. Schließlich haben wir den Umhang durch den Hintergrund ersetzt, sodass er im Bild „unsichtbar“ wird.