Aufgabenblatt 8

Einführung in die Bildverarbeitung
Christian Wilms und Simone Frintrop
SoSe 2020

Bei Fragen und Problemen schickt eine Mail an
wilms@informatik.uni-hamburg.de

Ausgabe: 26. Juni 2020 - **Abgabe bis: 3. Juli 2020, 10:00**

```
 Gelöste Aufgaben:
(_) Aufgabe 1.1
(_) Aufgabe 1.2
(_) Aufgabe 1.3
(_) Aufgabe 1.4
(_) Aufgabe 2.1
(_) Aufgabe 2.2
(_) Aufgabe 2.3
(_) Aufgabe 2.4
(_) Aufgabe 3.1
(_) Aufgabe 3.2
(_) Aufgabe 4.1
(_) Aufgabe 4.2
(_) Aufgabe 4.3
(_) Zusatzaufgabe 5.1
(_) Zusatzaufgabe 5.2
(_) Zusatzaufgabe 5.3
(_) Zusatzaufgabe 5.4
```

210	212	10	12	12
211	213	12	216	14
213	214	11	14	11
214	210	15	13	13
213	212	213	11	12
211	214	210	212	14
212	212	211	142	64
214	213	146	62	13
212	141	61	14	12

Abbildung 1: 8-bit Graustufenbild für Aufgabe 1 mit Pixelwerten als Zahlen.

Aufgabe 1 — Kanten finden mit Sobel und Prewitt - 5 + 5 + 3 + 7 = 20 Punkte - Theorieaufgabe
Gegeben sei für diese Aufgabe das Grauwertbild in Abbildung 1. Alle nachfolgenden Teilaufgaben sind händisch durchzuführen.

1. Wendet die beiden Prewitt-Operatoren auf die farbig markierten Pixel des Bildes an.
2. Wendet nun die beiden Sobel-Operatoren auf die farbig markierten Pixel des Bildes an und vergleicht das Ergebnis mit den Ergebnissen der Prewitt-Operatoren. Was verändert sich beim Wechsel von den Prewitt-Operatoren zu den Sobel-Operatoren?

3. Ermittelt aus den Ergebnissen der Sobel-Operatoren jeweils den Gradienten pro farbig markiertem Pixel.
4. Nutzt nun die Gradienten, um jeweils die Gradientenstärke sowie die Gradientenorientierung (in Grad) jedes farbig markierten Pixels zu ermitteln.

Aufgabe 2 — Kantenfindung und Störungen - 5 + 10 + 10 + 15 = 40 Punkte - Programmieraufgabe
Erlaubte (Sub-)Pakete: `numpy`, `skimage.io`, `skimage.filters`, `matplotlib`, `math`
Im Rahmen dieser Aufgabe soll die Erkennung von Kanten und der Effekt von Störungen darauf am Testbild `mandrill.png` genauer betrachtet werden.

1. Ladet euch das Testbild `mandrill.png` aus dem Moodle in Python. Wendet den Sobel-Filter auf das Bild an und lasst euch die Gradientenstärke pro Pixel als Ergebnisbild berechnen. Visualisiert dieses Bild. Wählt nun einen Grenzwert für die minimale Gradientenstärke und zeigt nur noch alle Pixel, die oberhalb dieses Grenzwerts liegen als Kante an (auf 1 setzen). Der Rest soll unterdrückt werden (auf 0 setzen). Beschreibt kurz das Ergebnis der Kantendetektion. Wurden die wichtigsten Kanten des Gesichts gefunden? Wurden zusätzlich weitere Kanten gefunden? Welcher Grenzwert führt zu einem sinnvollen Ergebnis?
   Hinweis: Obwohl es sich beim Ergebnisbild (dem Bild der Gradientenstärken) um ein Graustufenbild handelt, macht es Sinn, für die Visualisierung auf das Vorgeben der `cmap='gray'` zu verzichten. Dies führt zu einer Einfärbung des Ergebnisses, wobei hohe Werte gelbliche Farbtöne und niedrige Werte blaue Farbtöne bekommen.
2. Wendet nun vor der Anwendung des Sobel-Filters und der Berechnung der Gradientenstärken einen Gauß-Filter auf das Bild an. Experimentiert dabei mit verschiedenen σs. Ziel soll es dabei sein, dass möglichst nur noch die wichtigsten Kanten des Gesichts (Abgrenzungen der farbigen Nasenpartien, die Augen, Übergang von Nase zu Mund/Bart,...) erkannt werden. Wie verändert sich dabei das Ergebnis in Bezug auf die noch erkannten Kanten? Visualisiert das eurer Meinung nach beste Ergebnis erneut und benutzt dabei ebenso wieder einen Grenzwert, um nur recht starke Kanten zu visualisieren. Wie und warum verändert sich euer Grenzwert dabei mit steigendem σ?
   Hinweis: Setzt bei der Anwendung des Gauß-Filters den Parameter `preserve_range=True`, damit auch das Ergebnis im Bereich $0, \ldots 255$ bleibt.
3. Extrahiert nun die Bildzeile bei $x = 150$ aus dem Originalbild sowie aus der mit einem Gauß-Filter weichgezeichneten Variante aus der vorherigen Teilaufgabe. Fügt außerdem noch eine weitere Variante mit einem sehr hohen σ hinzu und extrahiert ebenfalls die genannte Bildzeile. Ermittelt nun eine Approximation der ersten Ableitung, indem ihr jeweils auf die gesamten Bilder den Sobel-Operator zur Detektion vertikaler Kanten anwendet. Extrahiert erneut die entsprechende Bildzeile bei $x = 150$ aus den Ergebnissen. Erstellt nun drei Plots mit der Funktion `matplotlib.pyplot.plot` (s. Aufgabenblatt 6) mit jeweils der extrahierten Bildzeile und der dazu passenden extrahierten Zeile des Ergebnisses des Sobel-Operators. Beschreibt kurz, wie sich die Kurven durch die unterschiedlichen Glättungen verändern.
   Tipp: Einen neuen Plot könnte ihr über die Funktion `matplotlib.pyplot.figure(i)` erzeugen, wobei `i` die laufende Nummer des Plots ist und bei 1 beginnt.
4. Berechnet nun die Orientierung (in Grad) des Gradienten in jedem Pixel mit Hilfe der Sobel-Operatoren auf dem Originalbild sowie auf eurem besten weichgezeichneten Ergebnis (Teilaufgabe 2). Erstellt anschließend jeweils ein Histogramm über die Gradientenorientierungen und visualisiert diese. Nutzt für das Histogramm 9 Behälter und setzt die Grenzen (`range`-Parameter) entsprechend. Welche Unterschiede gibt es zwischen den beiden Histogrammen? Wie ändert sich das Histogramm, wenn ihr den Parameter `weights` (den Parameter gibt es sowohl bei `numpy.histogram()` als auch bei `matplotlib.pyplot.hist()`) auf die entsprechenden Bilder mit den Gradientenstärken setzt?

Aufgabe 3 — Eigener Kantendetektor - 10 + 5 = 15 Punkte - Theorieaufgabe
Entwerft einen neuen Satz an Kantendetektoren. Diese Kantendetektoren sollen es ermöglichen, direkt Gradienten in den 8 Himmelsrichtungen einer Kompassrose zu erkennen. Die Himmelsrichtungen sind: Nord, Nordost, Ost, Südost, Süd, Südwest, West und Nordwest. Norden ist dabei im Bild oben, Osten rechts.

1. Gebt die acht 3×3 Operatoren, welche die entsprechend orientierten Gradienten in einem Bild erkennen. Achtet dabei darauf, dass ihr auch eine leichte Glättung einführt, um robuster gegen Rauschen zu sein. Nutzt als Gewichte in euren Operatoren Zahlen aus der folgenden Menge:

$\{-2, -1, 0, 1, 2\}$.

2. Auch die Sobel-Operatoren ermöglichen die Erkennung von Kanten, deren Gradienten in Richtungen wie Nordost oder Nordwest orientiert sind. Beschreibt kurz mit Hilfe eines kleinen Beispiels, wie dies funktioniert.

Aufgabe 4 — Farbkanten finden - 5 + 15 + 5 = 25 Punkte - Programmieraufgabe
Erlaubte (Sub-)Pakete: `numpy`, `skimage.io`, `skimage.filters`, `skimage.color`, `matplotlib`
Die Detektion von Kanten beschränkte sich bisher auf Grauwertbilder. Nun sollen auch Farbinformationen zur Ermittlung bestimmter Kanten, hier der Kante zwischen einem Gebäude und dem blauen Himmel, genutzt werden. Aus diesen Kanten kann in weiteren Schritte etwa eine stilisierte Skyline erzeugt werden.

1. Ladet euch zunächst das Bild `opera.png`, das einen Teil des Sydney Opera House vor einem blauen Himmel zeigt, aus dem Moodle in Python und wandelt es mit der Funktion `skimage.color.rgb2gray()` in ein Grauwertbild um. Ermittelt nun die Gradientenstärke je Pixel über die Sobel-Operatoren und visualisiert das Ergebnis. Welche Kanten werden gefunden und welche sind besonders stark?
2. Da wir nur an der Kante zwischen dem Gebäude und dem blauen Himmel interessiert sind, wollen wir nun die Farbinformationen des Bildes nutzen, um die Detektion von Kanten auf jene Kanten zu fokussieren. Dazu soll das RGB-Bild zunächst in einen anderen Farbraum umgewandelt werden, der Blau von anderen Farben und Graustufen trennt. Überlegt euch dazu einen Farb-„Raum“, eigentlich ist es nur eine Achse, in dem der Farbwert in etwa die Blauheit eines Farbtons wiedergibt. Konstruiert dazu eine Achse im RGB-Farbraum, die dies erfüllt und implementiert die Umwandlung des Bildes. Das Ergebnis ist wiederum ein Graustufenbild, da alle Farbwerte auf eine Achse abgebildet werden sollen, ähnlich der Achse der Graustufen im RGB-Farbraum. Allerdings müssen die Werte nicht notwendigerweise im Bereich $0, \ldots, 255$ liegen. Streckt den Kontrast des Bildes im neuen Farbraum abschließend so, dass das Minimum im Bild 0 ist und das Maximum 255.
   Tipp 1: Überlegt euch für den neuen Farbraum eine Achse, an deren einem Ende im RGB-Fabrraum Blau ist und an deren anderem Ende die Farbe im RGB-Farbraum ist, die am weitesten von Blau entfernt liegt.
   Tipp 2: Der neue Farbraum sollte im Bereich von $-255 \ldots 255$ liegen und alle Graustufen sollen den Wert 0 bekommen.
3. Erzeugt euch nun erneut die Gradientenstärke je Pixel über die Sobel-Operatoren und visualisiert das Ergebnis. Wie unterscheidet es sich vom vorherigen Ergebnis auf Basis der Graustufen?

Zusatzaufgabe 5 — Kanten ausdünnen - 10 + 15 + 10 + 5 = 40 Punkte - Programmieraufgabe
Erlaubte (Sub-)Pakete: `numpy`, `skimage.io`, `skimage.filters`, `matplotlib`, `math`
Ein Problem, das bei der bisherigen Kantendetektion auftrat, ist die Breite der Kanten. Die Breite der Kanten ist ein Problem, da Kanten im Bild meist idealerweise 1 Pixel breit sind, in den Ergebnissen der Kantendetektoren jedoch oft durch breitere Bereiche dargestellt werden. Für dieses Problem soll im Rahmen dieser Aufgabe eine Lösung erarbeitet werden, indem alle Pixel deren Gradientenstärke bezüglich einer Nachbarschaft nicht maximal ist, unterdrückt werden (auf 0 setzen). Die Nachbarschaft wird dabei durch die Orientierung der Gradienten bestimmt und verläuft orthogonal zur Kante.

1. Der Ausgangspunkt dieser Aufgabe ist das Ergebnis von Aufgabe 4, d.h. das Bild der Gradientenstärken auf Basis des neuen Farbraums, der die Blauheit darstellt. Ermittelt zusätzlich zu den Gradientenstärken die Orientierungen (in Grad) der Gradienten je Pixel. Fasst die Orientierungen o in vier Bereiche zu neuen Orientierungen o_{neu} zusammen:

$$o_{neu} \begin{cases} 0 & \text{falls } -22.5 \leq o < 22.5 \\ 45 & \text{falls } 22.5 \leq o < 67.5 \\ -45 & \text{falls } -67.5 \leq o < -22.5 \\ 90 & \text{sonst.} \end{cases}$$

   Das heißt, nach dieser Operation sollen alle Pixel den Wert $0, 45, -45$ oder 90 haben. Erzeugt zudem ein Ergebnisbild, das Größe und Datentyp des Bildes der Gradientenstärke haben soll, jedoch mit Nullen gefüllt ist.
2. Nun iteriert durch das Bild und ermittelt zunächst für jedes Pixel jene beiden Nachbarn aus der 3×3-Nachbarschaft, die orthogonal zur Kante (s. neu ermittelte Gradientenorientierung) zum

zentralen Pixel benachbart sind. Achtet dabei auch darauf, dass die Nachbarn im Bild existieren müssen und nicht außerhalb des Bildes liegen dürfen.

3. Ermittelt nun für jedes Pixel, ob die Gradientenstärke des Pixels größer oder gleich ist als jene seiner beiden in der vorherigen Teilaufgabe ermittelten Nachbarn. Ist dies der Fall, so wird für das Ergebnisbild die Gradientenstärke in diesem Pixel übernommen. Sollte dies nicht der Fall sein, wird die Gradientenstärke im Ergebnisbild auf 0 gesetzt. Visualisiert nun das Ergebnis, d.h. das Ergebnisbild mit den ausgedünnten Gradientenstärken. Hatte die Ausdünnung Erfolg?
4. Binarisiert nun das Ergebnis der ausgedünnten Gradientenstärken. Ist es möglich, einen Grenzwert zu wählen, sodass ein durchgehender Kantenzug, der die Kante zwischen Gebäude und Himmel darstellt entsteht? Falls nein, wie könnte man das auf Basis des Bildes der ausgedünnten Gradientenstärken ändern? Schlagt eine Lösung vor, implementieren müsst ihr sie nicht.