Aufgabenblatt 4

Einführung in die Bildverarbeitung
Christian Wilms und Simone Frintrop
SoSe 2020

Ausgabe: 22. Mai 2020 - **Abgabe bis: 29. Mai 2020, 10:00**

Aufgabe 0 — **Lehrevaluation Digitalisierung** - viel Ehre und Einfluss - Evaluationsaufgabe
Nehmt bitte bis Freitag, den 29. Mai 2020 an der Lehrevaluation zur Digitalisierung von

Übung Link zur Evaluation der Übung

und

Vorlesung Link zur Evaluation der Vorlesung

teil. Die Evaluationen ist selbstverständlich anonym und hilft uns die Veranstaltung zu verbessern, indem wir sie stärker an eure Bedürfnisse anpassen können.

(a) (b) (c)

Abbildung 1: Bilder für die Bildverbesserung in Aufgabe 1.

Aufgabe 1 — **Bildverbesserung** - 25 Punkte - Programmieraufgabe
Erlaubte (Sub-)Pakete: `numpy`, `skimage.io`, `matplotlib`, `math`
Abbildung 1 zeigt drei Bilder, die im Rahmen dieser Aufgabe mit Hilfe von jeweils einer Intensitätstransformation so verbessert werden sollen, dass eine vorgegebene Information einfacher aus den Bildern ausgelesen werden kann. Überlegt euch dazu zunächst welche Intensitätstransformation passend erscheint (mit kurzer Begründung) und implementiert diese in jeweils einer Python-Funktion. Die in Abbildung 1 dargestellten Bilder findet ihr im Moodle als `bildverbesserung.zip`.
Hinweis: Für die realistische Darstellung mit Hilfe von `matplotlib.imshow` nutzt die Parameter `vmin` und `vmax`. Diese Parameter sorgen dafür, dass die Grauwerte des Bildes nicht auf den Bereich 0 bis 255 skaliert werden. Diese Skalierung verzerrt den Kontrast künstlich, wenn ein Bild keine hohen oder niedrigen Grauwerte hat. Die Parameter können wie folgt genutzt werden:

```
import matplotlib.pyplot as plt
%für den Grauwertbereich 0 bis 1
plt.imshow(result,cmap='gray', vmin=0, vmax=1)
%oder für den Grauwertbereich 0 bis 255
plt.imshow(result,cmap='gray', vmin=0, vmax=255)
```

1. Verbessert die Sichtbarkeit des linken unteren Bildbereichs in Abbildung 1a und ermittelt, wie viele Poller es am Rande einer kleinen Grüninsel in der unteren linken Bildecke gibt?
2. Sorgt dafür, dass sich die Skyline in Abbildung 1b besser vom Himmel abhebt.

3. Verändert das Bild in Abbildung 1c so, dass in etwa der Bereich der Autobahn in der Bildmitte weiß ist und der Rest des Bildes seine Graufärbung behält.
   Hinweis: Das Ergebnis wird nicht perfekt werden!

Aufgabe 2 — Transformationsmatrizen - 30 Punkte - Theorieaufgabe
In dieser Aufgabe sollen Transformationen beschrieben oder interpretiert werden.

1. Gebt zu den drei Szenarien in Abbildung 2 die jeweilige Transformationsmatrix an. In den drei Szenarien sollen die Transformationen jeweils auf die Eckpunkte der Vierecke mit den roten Kreuzen angewendet werden, sodass jeweils das veränderte Vierecke (grüne Kreuze) entsteht. Bei Sequenzen von Transformationen sollen die einzelnen Transformationsmatrizen und die Berechnung der finalen, kombinierten Transformationsmatrix gegeben werden.
   Tipp: Wendet einmal eure erzeugte Transformationsmatrizen auf die entsprechenden Punkte an und betrachtet das Ergebnis. Stimmt es?
2. Interpretiert die drei folgenden Transformationsmatrizen und beschreibt kurz welche einzelnen Transformationen (inklusive Parameter wie Rotation um 45°) in welcher Reihenfolge hier beschrieben werden.
   Tipp: Es gibt tlw. mehrere Lösungen.
   a)
   $$\begin{bmatrix} 0.981 & -0.195 & 0 \\ 0.195 & 0.981 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$
   b)
   $$\begin{bmatrix} 0.707 & 0.707 & 0.707 \\ -0.707 & 0.707 & 3.536 \\ 0 & 0 & 1 \end{bmatrix}$$
   c)
   $$\begin{bmatrix} 1 & 0 & 3 \\ -8 & -4 & -2 \\ 0 & 0 & 1 \end{bmatrix}$$

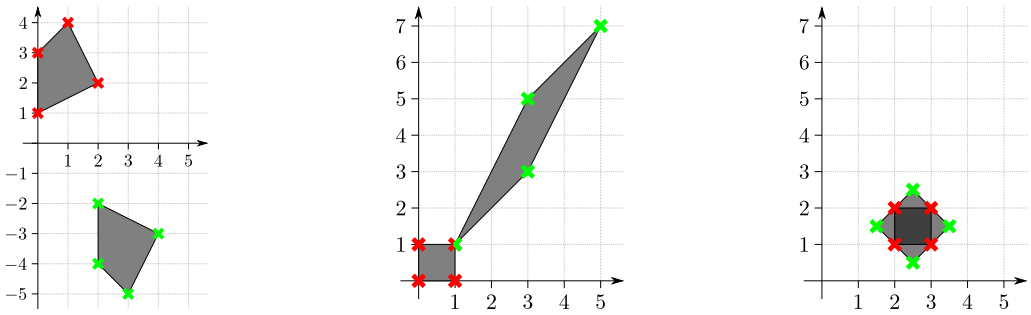

Abbildung 2: Drei Szenarien, in denen jeweils eine Transformationsmatrix auf die Eckpunkt eines Vierecks (rote Kreuze) angewendet wurde, um ein neues Viereck zu erzeugen (grüne Kreuze).

Aufgabe 3 — Inverse Abbildung - 15 Punkte - Theorieaufgabe
Bei der Implementation von Bildtransformationen wird meist, wie in der Vorlesung besprochen, die Inverse der Transformationsmatrix (inverse Mapping) genutzt, um Bildpunkte aus dem Zielbild zurück in das Ursprungsbild zu transformieren. An diesen Stellen im Ursprungsbild wird dann die Interpolation durchgeführt. Dafür müssen die Inversen zunächst bestimmt werden. Führt dies an den folgenden drei Transformationsmatrizen durch und bildet die inversen Transformationsmatrizen. Nutzt dazu beispielsweise den Gauß-Jordan-Algorithmus.
Hinweis: Ohne Rechenwege gibt es hier keine Punkte.

1.
$$\begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

2.
$$\begin{bmatrix} -2\cos(\pi) & 2\sin(\pi) & 2 \\ \sin(\pi) & -\cos(\pi) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

3.
$$\begin{bmatrix} 1 & 0 & 2 \\ 2 & 3 & 4 \\ 0 & 0 & 1 \end{bmatrix}$$

Aufgabe 4 — Bildskalierung und Interpolation - 30 Punkte - Programmieraufgabe
Erlaubte (Sub-)Pakete: `numpy`, `skimage.io`, `matplotlib`, `itertools`
Durch die geometrische Transformation von Bildern, bspw. die uniforme Skalierung um einen Faktor s, werden Pixelwerte an Koordinaten des Ursprungsbildes benötigt, die zwischen den eigentlichen Koordinaten liegen. So kann es etwa vorkommen, dass ein Wert an der Koordinate $(0.2, 0.8)$ benötigt wird. Da es in Bildern aber nur ganzzahlige Koordinaten gibt, muss der Pixelwert an der Koordinate $(0.2, 0.8)$ interpoliert werden. Dies soll in dieser Aufgabe mit der nearest neighbor Interpolation im Rahmen der Bildskalierung umgesetzt werden. Bei der nearest neighbor Interpolation wird, wie in Abbildung 3a dargestellt, der Pixelwert der Koordinate genommen, die der benötigten Koordinate (bspw. $(0.2, 0.8)$) am nächsten liegt. In diesem Beispiel wäre dies die Koordinate $(0, 1)$.

1. Schreibt eine Python-Funktion, die ein Bild um einen gegebenen Faktor, der nicht ganzzahlig sein muss, uniform skaliert. Nutzt dazu die nearest neighbor Interpolation.
   Hinweis 1: Rechnet die Koordinaten des Zielbildes in die Koordinaten des Ausgangsbildes zurück und interpoliert dort.
   Hinweis 2: Die Funktion `itertools.product()` kann nützlich sein, um aus den einzelnen x- bzw. y-Koordinaten alle Koordinaten des Bildes zu erzeugen.
2. Testet nun eure Skalierung am Bild `tv.png` aus dem Moodle.
3. Werden durch das Skalieren von Bildern mit einem Faktor $s > 1$ Informationen gewonnen?
4. Was passiert, wenn ihr das Bild zunächst um die Hälfte verkleinert ($s = 0.5$) und anschließend wieder auf die vorherige Größe vergrößert ($s = 2$)? Ist das Ergebnis identisch zum Ursprungsbild?

(a) Nearest neighbor Interpolation

(b) Bilineare Interpolation

Abbildung 3: Vergleich von nearest neighbor Interpolation und bilinearer Interpolation. Der Grauwert an der schwarzen Koordinate, die nicht im Bild existiert, soll interpoliert werden. Die anderen vier Koordinaten existieren im Bild und sind die Basis für die jeweilige Interpolation. Quelle: Wikimedia [1]

Zusatzaufgabe 5 — Bilineare Interpolation - 30 Bonuspunkte - Programmieraufgabe
Erlaubte (Sub-)Pakete: `numpy`, `skimage.io`, `matplotlib`, `itertools`
Verändert die Implementation aus Aufgabe 4 so, dass statt der nearest neighbor Interpolation die bilineare Interpolation [2] genutzt wird. Bei der bilinearen Interpolation wird, wie in Abbildung 3b visualisiert, in zwei Schritten linear zwischen allen vier umgebenen Koordinaten interpoliert. Im oben genannten Beispiel für die Koordinate $(0.2, 0.8)$ also zwischen den Koordinaten $(0, 0)$, $(0, 1)$, $(1, 0)$ und $(1, 1)$. Zunächst wird in

[1] https://commons.wikimedia.org/wiki/File:Comparison_of_1D_and_2D_interpolation.svg
[2] https://en.wikipedia.org/wiki/Bilinear_interpolation

x-Richtung linear interpoliert (in Abbildung 3b von links nach rechts), damit anschließend in y-Richtung (in Abbildung 3b von vorne nach hinten) zwischen den vorher interpolierten Werten erneut interpoliert werden kann.

1. Fügt eurer Skalierungsfunktion aus Aufgabe 4 nun einen Parameter hinzu, der die Interpolation von nearest neighbor auf bilineare Interpolation umschalten kann und implementiert diese Interpolation. Nutzt dazu bspw. die Beschreibung von Wikipedia.
   Hinweis 1: Für die vier benachbarten Koordinaten sucht euch erstmal nur die Koordinate, deren x- und y-Wert kleiner ist. Daraus könnt ihr durch Addition von 1 die anderen drei Koordinaten gewinnen.
   Hinweis 2: Achtet darauf, dass die zur Interpolation benötigten Koordinaten alle im Bereich des Ursprungsbildes liegen müssen oder entsprechend verändert werden müssen.
   Hinweis 3: Am rechten und unteren Seitenrand kann es zu schwarzen Streifen kommen. Dies liegt daran, dass dort die benachbarten Koordinaten tlw. gleich sind. Erzwingt hier eine Änderung.
   Hinweis 4: Bei der Interpolation in Bildern gilt meist, dass die benachbarten Koordinaten zwischen denen interpoliert wird, einen Abstand von 1 entlang der x- bzw. y-Achse haben. Daher lassen sich die Formeln der bilinearen Interpolation vereinfachen.
2. Testet auch die Skalierung mit bilinearer Interpolation am Bild `tv.png` aus dem Moodle. Welche Unterschiede erkennt ihr in den Ergebnissen?