Aufgabenblatt 3

Einführung in die Bildverarbeitung
Christian Wilms und Simone Frintrop
SoSe 2020

Ausgabe: 15. Mai 2020 - **Abgabe bis: 22. Mai 2020, 10:00**

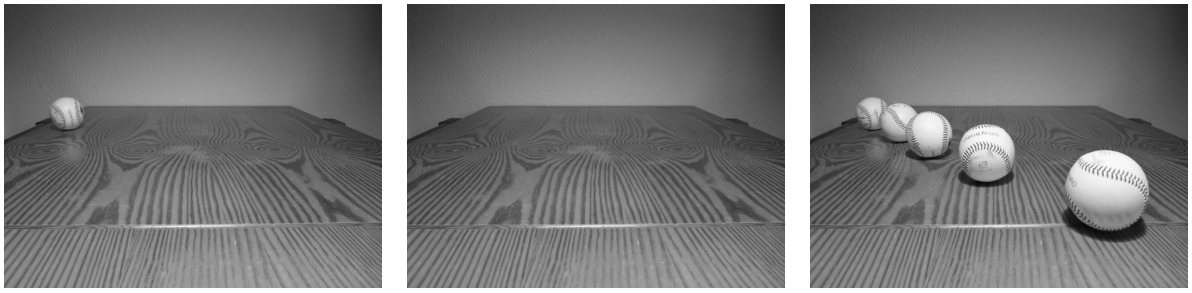
Aufgabe 1 — Gesetz der großen Zahlen - 30 Punkte - Theorieaufgabe
In Aufgabe 4 des Aufgabenblatts 2 solltet ihr bereits den Zusammenhang $\mathrm{E}(\bar{\mathrm{g}}(x,y)) = \mathrm{f}(x,y)$ im Rahmen des Gaußschen Rauschens beweisen. Nun soll im selben Szenario der Zusammenhang für die Varianzen betrachtet werden. Beweist daher, dass

$$\sigma^2_{\bar{\mathrm{g}}(x,y)} = \frac{1}{M}\sigma^2_{\eta(x,y)}$$

gilt, wobei $\sigma^2_{\bar{\mathrm{g}}(x,y)}$ die Varianz von $\bar{\mathrm{g}}(x,y)$ ist und $\sigma^2_{\eta(x,y)}$ die Varianz von $\eta(x,y)$. Den Rest des Szenarios entnehmt ihr bitte Aufgabe 4 des Aufgabenblatts 2.

Aufgabe 2 — Serienbild - 20 Punkte - Programmieraufgabe
Erlaubte (Sub-)Pakete: `numpy`, `skimage.io`, `matplotlib`
In der Vorlesung wurde das Mitteln über mehrere zeitlich versetzt aufgenommene Bilder einer identischen Szene als Möglichkeit zur Unterdrückung des Rauschens präsentiert. Diese Idee soll in dieser Aufgabe genutzt werden, um die Einzelbilder eines Serienfotos (bspw. Abbildung 1a) zu einem Bild, wie in Abbildung 1c zu sehen ist, zusammenzusetzen. Dabei zeigt das Serienfoto die Bewegung eines Balls über eine Tischplatte. Das Endergebnis soll alle fünf Stadien der Bewegung aus den Einzelbildern auf einem Bild vereinen.

1. Ladet die fünf Einzelbilder des `serienbild.zip` aus dem Moodle in Python. Zunächst muss aus den fünf Einzelbildern ein möglichst perfektes Hintergrundbild ohne Ball erzeugt werden, ähnlich dem in Abbildung 1b. Wenn man den Ball als Rauschen annimmt, kann man nach dem Gesetz der großen Zahlen durch Mittelung über die Bilder das Rauschen, bzw. hier den Ball, zumindest recht gut entfernen. Reichen fünf Bilder dafür aus? Probiert es aus!
2. Überprüft das Ergebnis visuell und überlegt euch eine bessere aber sehr ähnliche Möglichkeit für das Verknüpfen der fünf Pixel an jeweils einer Koordinate. Betrachtet dabei keine anderen Koordinaten wie etwa die der Nachbarn! Das Ergebnis soll dabei dem in Abbildung 1b so nahe wie möglich kommen.
3. Nutzt nun euer erzeugtes Hintergrundbild, um die veränderten Pixel in jedem der fünf Einzelbilder zu ermitteln. Damit ihr wirkliche Veränderungen vom Grundrauschen unterscheiden könnt, empfiehlt sich die Nutzung eines Schwellenwerts für die Mindeststärke der Veränderung. Den Schwellenwert müsst ihr selbst ermitteln. Ein perfektes Ergebnis werdet ihr aber vermutlich nicht bekommen, wie auch im Ergebnisbild 1c zu erkennen ist.
4. Ersetzt nacheinander für jedes der fünf Bilder die veränderten Pixel im Hintergrundbild durch die entsprechenden Pixel des Einzelbildes. D.h. die Pixel, die sich zwischen dem Hintergrundbild und dem ersten Einzelbild verändert haben, werden im Hintergrundbild durch die entsprechenden Pixel des ersten Einzelbilds ersetzt usw.
5. Speichert das Ergebnisbild ab. Es sollte in etwa dem in Abbildung 1c entsprechen.
   Tipp: Nehmt selbst ein Serienfotos oder Einzelfotos auf und versucht euer Verfahren darauf anzuwenden. Achtet dabei auf eine möglichst ruhige Kamera und benutzt wenn möglich ein Stativ. Um ein Farbbild in ein Graustufenbild umzuwandeln, könnt ihr die Funktion `skimage.color.rgb2gray` nutzen und das Ergebnis mit 255 multiplizieren. Originelle Ergebnisse werden in den Lösungsvideos gezeigt!



(a) (b) (c)

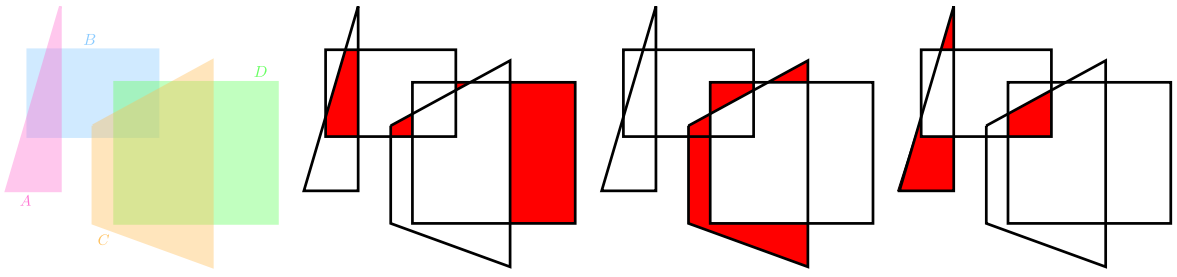
Abbildung 1: Erstes Einzelbild (a) eines Serienbilds, aus den Einzelbildern berechnetes Hintergrundbild (b) und Überlagerung der fünf Einzelbilder (c).

Aufgabe 3 — Findet die Fehler - 25 Punkte - Programmieraufgabe
Erlaubte (Sub-)Pakete: `numpy`, `skimage.io`, `matplotlib`
Das Finden von Fehlern in der manipulierten Variante eines Originalbilds ist ein beliebtes Rätsel. Im Rahmen dieser Aufgabe soll dieses Problem nun strukturiert angegangen und mithilfe eines Python-Skripts gelöst werden. Dazu findet ihr im Moodle zwei Varianten eines Bilds, die eine ohne Veränderungen, die andere mit Veränderungen.

1. Ermittelt zunächst ein Bild, das die Veränderungen zwischen den beiden Bildern zeigt als Verküpfung der Bilder mit einem sinnvollen arithmetischen Operator.
2. Wandelt das Ergebnis anschließend in ein Binärbild um, das veränderte Pixel als Vordergrund und unveränderte Pixel als Hintergrund beinhaltet. Wenn ihr euch das Bild nun anzeigen lasst, könnt ihr zwar die Positionen der Veränderungen selbst ermitteln und die Anzahl zählen, aber das soll nun ebenfalls der Computer machen.
3. Erweitert daher euer Skript, um die Funktionalität des Zählens der Veränderungen. Dabei ist jeder zusammenhängende Bereich an Pixeln, der sich verändert hat, als ein Bereich zu zählen.
   a) Ermittelt dazu zunächst alle Koordinaten von Pixeln, die sich verändert haben.
   b) Startet nun bei einem beliebigen Pixel und prüft ob dessen Nachbarn auch verändert wurden. So könnt ihr euch durch die Menge der Vordergrundkoordinaten hangeln und iterativ die zusammenhängenden, veränderten Bereich vollständig aufspüren.
   c) Welche Anzahl an veränderten Bereichen ermittelt euer Skript?

Hinweis 1: Nutzt nur die oben angegebenen Pakete und Subpakete, sonst gibt es keine Punkte!
Hinweis 2: Achtet beim Durchlaufen der Vordergundkoordinaten auf sinnvolle Abbruchkriterien, um endlose Schleifen zu vermeiden.

Aufgabe 4 — Mengenoperationen auf Bildern - 10 Punkte - Theorieaufgabe
In Abbildung 2a ist ein Bild mit vier markierten Bildbereichen dargestellt. Jeder der vier farbigen Bildbereiche (A, B, C, D) entspricht einer Menge an Koordinaten, die überdeckt werden. Gebt nun mithilfe der Mengenoperationen die Terme an, die den rot markierten Bereichen in den Bildern 2b, 2c und 2d entsprechen (ein Term pro Bild für jeweils alle roten Bereich zusammen).



(a) (b) (c) (d)

Abbildung 2: Vier Koordinatenmengen (a) und drei verschiedene Anwendungen (b), (c) und (d) von Mengenoperationen auf diese.

Aufgabe 5 — Räumliche Operationen - 15 Punkte - Theorieaufgabe
Schaut euch die acht folgenden Python-Funktionen an und beantwortet zu jeder Funktion, ob es sich dabei um eine Punktoperation, eine Nachbarschaftsoperation, eine geometrische Transformation oder eine globale Operation handelt. Gebt jeweils eine kurze Begründung.
Hinweis: Die Nachbarschaft eines Pixels muss nicht unbedingt alle bzw. nur die acht direkt benachbarten Pixel enthalten.

```
def a(img):
    return img+1

def b(img):
    result = np.copy(img)
    for x in range(img.shape[0]):
        for y in range(1,img.shape[1]):
            result[x,y-1] = img[x,y-1]
    return result

def c(img):
    result = np.zeros_like(img)
    result[:,:] = np.max(img)
    return result

def d(img):
    result = np.zeros_like(img)
    for x in range(img.shape[0]):
        for y in range(img.shape[1]):
            result[x,y] = img[int(x*cos(pi)-y*sin(pi)),int(x*sin(pi)+y*cos(pi))]
    return result

def e(img):
    return img[:,1:]-img[:,:-1]

def f(img):
    return img-np.full(img.shape,np.mean(img))

def g(img):
    result = np.copy(img)
    for x in range(img.shape[0]):
        for y in range(img.shape[1]-1):
            a = img[x,y]
            b = img[x,y+1]
            if True:
                result[x,y]=a
            else:
                result[x,y]=b
    return result

def h(img):
    result = np.zeros_like(img)
    for x in range(img.shape[0]):
        for y in range(img.shape[1]):
            result[x,y] = np.mean(img[:,y])
    return result
```