Bildverarbeitung - Übungsblatt 2

Assistenten: Binbin Xiang, Shengyu Huang, Nadine Rüegg
Hauptkontakt: binbin.xiang@ethz.ch
Datum: April 2022

Für das zweite Übungsblatt zur Vorlesung Bildverarbeitung sind einige Bildverarbeitungsaufgaben beispielhaft zu implementieren. Reichen Sie Ihren Code ein, sowie einen kurzen Bericht, der für jede Teilaufgabe die Resultate und falls notwendig einen kurzen Kommentar dazu enthält ($\leq$4 Sätze pro Teilaufgabe, auf Englisch).
Dieses Übungsblatt trägt 80% zur Gesamtnote für den Kurs bei. Reichen Sie Ihre Lösungen zu diesem Aufgabenteil bis spätestens am 26. Mai 2022 ein. Jede/r Studierende bespricht seine/ihre Antworten individuell mit den Assistierenden. Diese Abgaben finden statt am 31. Mai 2022 zwischen 15:45 und 17:30.
Die vorbereitete Codebasis finden Sie hier: https://github.com/prs-eth/Bildverarbeitung-FS2022. **Es ist nicht erlaubt, andere Bibliotheken zu verwenden als jene, die dort bereits importiert sind!**

1 Fouriertransformation

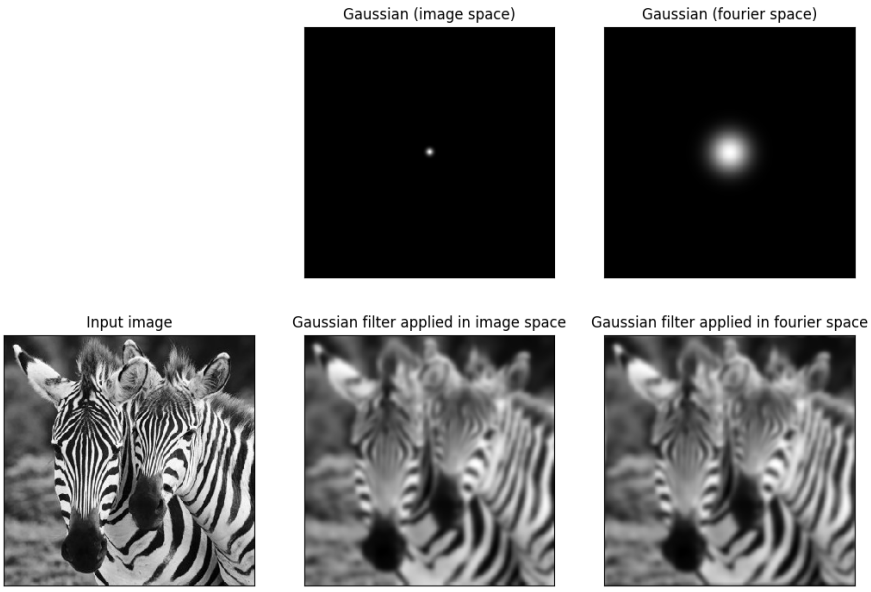
Im folgenden werden Sie mit der *Fast Fourier Transform (FFT)* arbeiten, einem effizienten Algorithmus zur Berechnung der *Diskreten Fourier Transform (DFT)*. Füllen Sie *fourier.ipynb* aus.

1.1 Transformation

Laden Sie das Inputbild *tuebingen.jpg*, transformieren Sie dieses mit der FFT (*scipy.fft.fft2*) und plotten Sie das Spektrum. In diesem und allen folgenden Plots sollen tiefe Frequenzen in der Bildmitte geplottet werden und hohe am Rand des Bildes. Verwenden Sie dazu *scipy.fft.fftshift* und *scipy.fft.ifftshift*. Hinweise: (i) Plotten Sie nur die Magnitude des resultierenden Spektrums, nicht die Phase. (ii) Logarithmische Skalierung der Helligkeiten verbessert den Kontrast der Visualisierung.

1.2 Hochpass-Filter

Filtern Sie das Spektrum nun so, dass tiefe Frequenzen unterdrückt werden. Implementieren sie eine Funktion *highpass_filter*, die den idealen Hochpassfilter



(a) Bild 1

Figure 1: Fouriertransformation: Beispielresultat

(harte Kreisblende, Radius 30 Pixel) retourniert. Wenden Sie den Filter im Frequenzraum an. Transformieren Sie das gefilterte Spektrum zurück in den Ortsraum und plotten Sie das resultierende Bild sowie das gefilterte Spektrum.

1.3 Gaussfilter

Ziel dieser Teilaufgabe ist es, einen Gauss'schen Tiefpassfilter sowohl im Orts- als auch im Frequenzraum anzuwenden. Bis auf numerische Ungenauigkeiten sollten Sie identische Resultate erhalten.

1.3.1 Konstruktion eines Gaussfilters

Definieren Sie die Funktion *g_kern*, die einen Gauss-Kern retourniert. Verwenden Sie hierzu die Dichtefunktion *scipy.stats.norm.cdf*, um für jedes Intervall (Pixel) das korrekte Filtergewicht zu berechnen.

1.3.2 Faltung mit dem Gaussfilters

Konstruieren Sie einen Gaussfilter mit Standardabweichung 5. Setzen Sie zunächst die Filtergrösse mit der Bildgrösse gleich. Da die Werte weit weg von der Filtermitte sich kaum von 0 unterscheiden, können sie für die Faltung ein kleineres

Fenster (ca. 6× Standardabweichung, ungerade Seitenlänge in Pixel) ausschneiden und dieses für die Faltung mit dem Inputbild verwenden. Verwenden Sie hierzu *scipy.ndimage.convolve*.

1.3.3 Transformation des Filters in den Frequenzraum

Transformieren sie den Filter in den Frequenzraum. Wie in Aufgabe 1.1 beschrieben, wollen wir auch hier die tiefen Frequenzen in der Bildmitte haben, ordnen Sie zu diesem Zweck sowohl die Frequenzdarstellung des Inputbildes als auch die Frequenzdarstellung des Gaussfilters mit Hilfe von *scipy.fft.fftshift* um.

1.3.4 Faltung mit Gaussfilter im Frequenzraum

Multiplizieren Sie die Fouriertransformierte des Bildes mit dem transformierten Gaussfilter (nur Magnitude). Transformieren Sie das Resultat zurück in den Ortsraum.

1.3.5 Visualisierung

Visualisieren Sie:

- Den Gaussfilter im Ortsraum
- Den Gaussfilter im Frequenzraum
- Das Outputbild nach Faltung mit dem Gaussfilter im Ortsraum
- Das Outputbild nach Multiplikation mit dem Gaussfilter im Frequenzraum

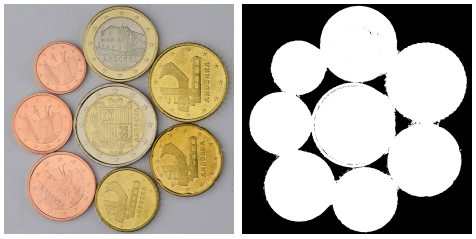
Figure 1 zeigt ein Beispiel (anderes Bild, andere Filterbreite) wie Ihre Resultate dargestellt werden sollten. Berechnen und visualisieren Sie zusätzlich die Abweichung zwischen den beiden Resultaten.

1.3.6 Separierbare Filterung

In Teilaufgabe 1.3.1 haben Sie einen 2D Gaussfilter benutzt. Wiederholen Sie diese Teilaufgabe, wobei diesmal die Filterung getrennt in x- und $y-$Richtung durchgeführt wird. Vergleichen Sie die Ergebnisbilder und die Rechenzeit. Hinweis: um eine zuverlässige Messung der Rechenzeit zu erhalten ist es nötig, die Filterung für ein grosses Bild oder mehrmals zu berechnen.

2 Morphologie

In dieser Aufgabe wird die grafische Benutzeroberfläche (GUI) verwendet, und Sie werden im folgenden die Funktionen in *libs/morphology.py* implementieren.



(a) Münzen (b) Segmentierung

Figure 2: Ausgangslage Münzen-Aufgabe

2.1 Münzen

Wir stellen ein Inputbild *coins.jpg* und eine einfache, ungefähre Segmentierung *coins_seg.png* zur Verfügung.

2.1.1 Segmentierung

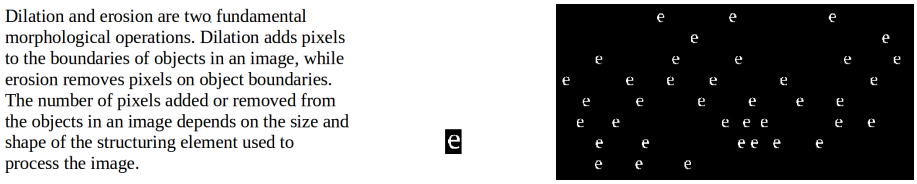
Segmentieren sie das Inputbild mit den Münzen *coins.jpg* um eine Segmentierung in Münzen und Hintergrund zu erhalten. Wie Sie diese Aufgabe lösen, können Sie selbst wählen. Experimentieren Sie mit verschiedenen Segmentierungs-/Clusteringmethoden und Farbräumen. Versuchen Sie, mindestens die Qualität der Maske in *coins_seg.png* zu erreichen.

2.1.2 Schliessen der Löcher innerhalb der Münzen

Starten Sie mit dem bereitgestellten Segmentierungsbild *coins_seg.png* und benutzen Sie *cv2.dilate* und *cv2.erode* um die Löcher innerhalb der einzelnen Münzen zu schliessen. Visualisieren Sie das Resultat.

2.1.3 Instanzsegmentierung von Münzen

Starten Sie mit dem Resultat der letzten Teilaufgabe und erweitern Sie dieses so, dass die Münzen einzeln segmentiert werden. Hinweis: Sie können markerbasierte Segmentierung mithilfe des Watershed Algorithmus verwenden. Verwenden Sie als erstes morphologische Operationen und die Funktion *cv.connected Components* um angeschlossene Komponenten als Marker für den Watershed Algorithmus zu finden. Es ist ausreichend eine Komponente pro Münze zu finden, die Form muss nicht exakt mit der Form der Münze überrein stimmen. Lesen Sie in einem zweiten Schritt das initiale Bild *coins.jpg* und wenden sie den Watershed Algorithmus an (*cv.watershed*). Generieren Sie in einem letzten Schritt zufällige Farben und färben Sie die einzelnen Münzen ein. Auch der Hintergrund soll eingefärbt werden und sich von den Münzen abheben.



(a) Input-Text (b) Buchstabe (c) Resultat

Figure 3: Buchstabenerkennung

2.2 Buchstaben-Erkennung

Bei dieser Aufgabe besteht der Input aus einem gerasterten Text (Fig. 3a) sowie einer Vorlage für den Buchstaben "m" (Fig. 3b). Benutzen sie morphologische Operationen (Dilatation, Erosion, wenn nötig weitere) um aus dem Text alle Buchstaben "m" zu extrahieren und ein Resultatbild wie das in Fig. 3c zu erzeugen.

3 Eckendetektor

Implementieren Sie die Eckendetektoren von Harris und von Shi-Tomasi für Schwarzweissbilder. Folgen Sie hierzu den Instruktionen in *corner.ipynb* und füllen Sie den *TODO*-Teil aus. Visualisieren Sie alle Zwischenresultate mit Hilfe des zur Verfügung gestellten Skriptes. Die gewünschten Zwischenergebnisse sind für jede der beiden Methoden:

- Harris / Shi-Tomasi Antwort pro Pixel
- Pixel über dem gewählten Ecken-Schwellwert
- Ecken nach Non-Maximum-Suppression

Wenden Sie denselben Detektor auf rotierte Versionen des Bildes an. Vergleichen Sie die Ergebnisse im Bericht.

4 Canny Kantendetektion

Implementieren Sie einen Kantenpixel-detektor für Grauwertbilder. Folgen Sie hierzu den Instruktionen in *canny_edge.ipynb* und füllen Sie den *TODO*-Teil aus. Visualisieren Sie Zwischenresultate mit Hilfe des zur Verfügung gestellten Skriptes:

- Input
- Bild nach Anwendung des Gaussfilters
- Beträge und Richtungen der Gradienten

- Canny-Kantenpixel nach Non-Maximum-Suppression

Kommentieren Sie im Bericht Gemeinsamkeiten und Unterschiede zur Eckendetektion.

5 Bonusaufgabe

Integrieren Sie eine der drei Aufgaben aus den Jupyter Files in die grafische Benutzeroberfläche (GUI). Sie können diese Ihren Bedürfnissen nach anpassen.