Zwischenbericht Master-Projekt Bildverarbeitung

SoSe 2015

Dorothee Geiser, Niels Porsiel

27. Juli 2015

Im laufe des Semesters haben wir uns mit Scale-Invariant Feature Transform (kurz SIFT) beschäftigt. Dabei haben wir uns am Paper *Distinctive Image Features from Scale- Invariant Keypoints* von David G. Lowe [Lowe04] orientiert. Unsere Aufgabe war es zu versuchen, SIFT nachzubauen. Dieser Zwischenbericht soll nun einen Überblick darüber geben, was wir geschafft haben.

Inhaltsverzeichnis

1 Unser Code

1.1 Keypoint-Erkennung

Die erste Aufgabe war das Erkennen von Keypoints in einem Grauwertbild. Gesucht sind prominente Hell-/Dunkel-Kontraste, die (skaleninvariant) wiederzufinden sein sollen. Um dies zu bewerkstelligen, bauen wir, nach Vorgabe von Kapitel 3 von Lowe [Lowe04], ein Set aus Oktaven mit je fünf Skalen auf. Die Anzahl Oktaven ergibt sich dabei aus dem binären Logarithmus des Minimums von Höhe und Breite des Ausgangsbildes minus drei (= die maximale Anzahl mit der das Bild um den Faktor vier verkleinert werden kann). Bevor die Oktaven erzeugt werden, muss festgelegt werden, ob das Ausgangsbild auf die doppelte Größe skaliert wird um eine extra Oktave zu erzeugen. Dies geschieht bei uns mittels des boolschen Parameters `double_image_size`, der dem Programm bei Aufruf übergeben werden kann (default ist true).

Zum Aufbau der Oktaven wird das (eventuell skalierte) Ausgangsbild mithile von einem Gaußfilter gefaltet und diese Operation so oft wiederholt, bis die festgelegte Anzahl Bilder pro Oktave (bei uns fünf) erreicht ist. Das letzte und unschärfste Bild einer Oktave wird skaliert, so dass es nur noch die Hälfte in Höhe und Breite misst, also um den Faktor 4 verkleinert und als Ausgangsbild für die nächste Oktave verwendet.
Die Berechnung der Standardabweichung des Gaußfilters kann dabei iterativ oder absolut durchgeführt werden. Bei der iterativen Berechnung wird die Standardabweichung abhängig von der Standardabweichung des Bildes aus der letzten Iteration mittels der Formel $\sqrt{\sigma_{total}^2 - \sigma_{last}^2}$ berechnet, wobei $\sigma_{last} = k^{(i-1)} * \sigma$ und $\sigma_{total} = \sigma_{last} * k$. k wird im Paper von Lowe als $k = 2^{(1/s)}$ definiert, für s haben wir 2 als Wert gewählt, i ist die Iteration in der aktuellen Oktave und σ kann auch als Aufrufparameter an das Programm übergeben werden (default ist 1.0). Die absolute Methode verwendet zur Berechnung immer das Ausgangsbild der aktuellen Oktave. Die Standardabweichung berechnet sich hierbei durch die Formel $k^i * \sigma$. Die Wahl der Methode wird über die boolsche Variable `iterative_interval_creation` gesteuert und kann dem Programm auch beim Aufruf übergeben werden (default ist true).

Aus zwei benachbarten Bildern jeder Oktave wird jeweils ein sogenannter Difference-of-Gaussian (DOG) gebildet. Dafür wird die Differenz der Grauwerte in jedem Pixel errechnet und aus den resultierenden Werten ein neues Bild erzeugt. Sobald mindestens drei DOGs vorhanden sind, wird in diesen nach Extrempunkten gesucht. Dies geschieht, indem der Grauwert jedes Pixels mit dem seiner umliegenden Nachbarn verglichen wird; 8 direkte Nachbarn im eigenen DOG und jeweils 9 im darüber und darunter liegenden (siehe Abbildung 2 im Paper von Lowe). Pixel, die am Rand des aktuellen DOGs oder im ersten / letzten DOG liegen, werden dabei ignoriert.
Ist der Wert des Pixels höher oder niedriger als der aller seiner Nachbarn und größer als ein festgelegter Grenzwert `dog_threshold` (Aufrufparameter, default ist 5.0), wird er als Keypoint erkannt. Der Vergleich wird in der Inline Funktion `localExtremum` durchgeführt, die als Parameter einen Pointer auf einen Standardvektor `dog` aus vigra-

Multiarrays und drei Integer i, x und y für den aktuellen Pixel annimmt. Dabei sind x und y die Koordinaten des Pixels im DOG und i der Index für den DOG-Vektor, in welchem sich der Pixel befindet. Die Funktion liefert den boolschen Wert true zurück, wenn der Pixel ein Extrempunkt ist (sonst false).

Die so erkannten Keypoints sieht man in Abbildung 1.

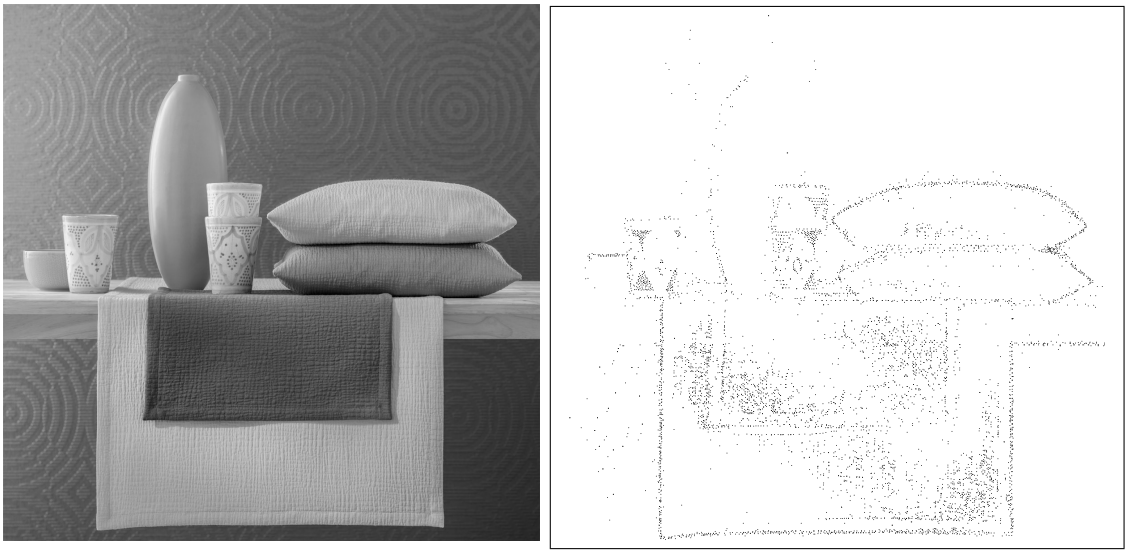

Abbildung 1: Keypoint-Erkennung. Links das Original, rechts die erkannten Keypoints.

1.2 Keypoint-Reduktion und Verschiebung auf Subpixel-Ebene

Im nächsten Schritt sollte die Anzahl der erkannten Keypoints reduziert werden und [...]. Das Ergebnis sieht man in Abbildung 2.

1.3 Keypoint-Orientierung

Dann folgte die Keypoint-Orientierung. Das Ergebnis sieht man in Abbildung 3.

1.4 Keypoint Deskriptoren

Schlussendlich kamen noch die Keypoint Deskriptoren.

2 Vergleich des Erreichten mit der Vorgabe

Im Vergleich mit dem, was Lowe [Lowe04] getan hat ...

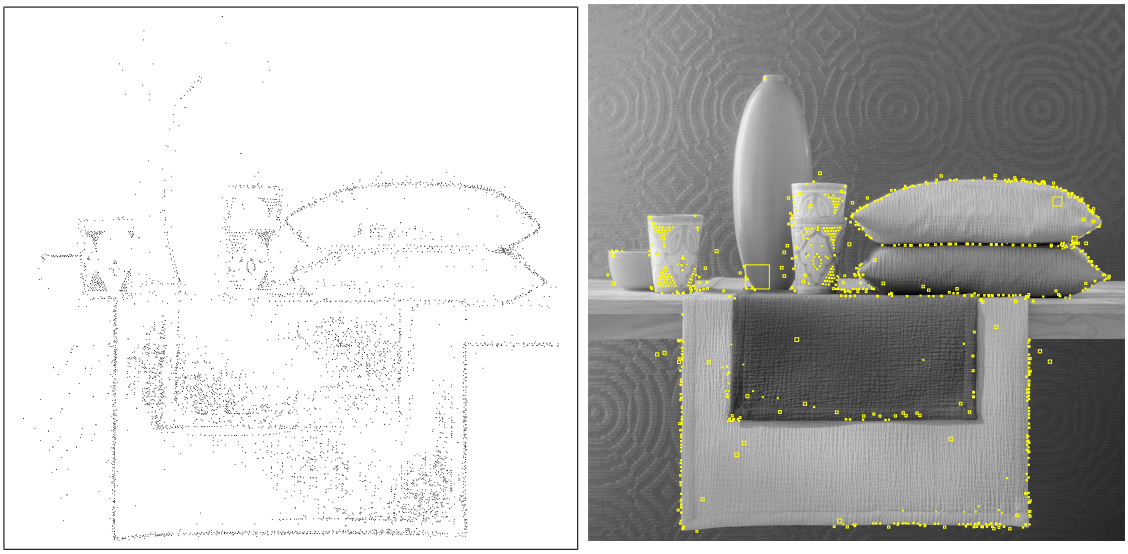

Abbildung 2: Keypoint-Reduktion und Verschiebung auf Subpixel-Ebene. Links die vorher erkannten Keypoints, rechts die jetzt erkannte, [...] geringere Menge.

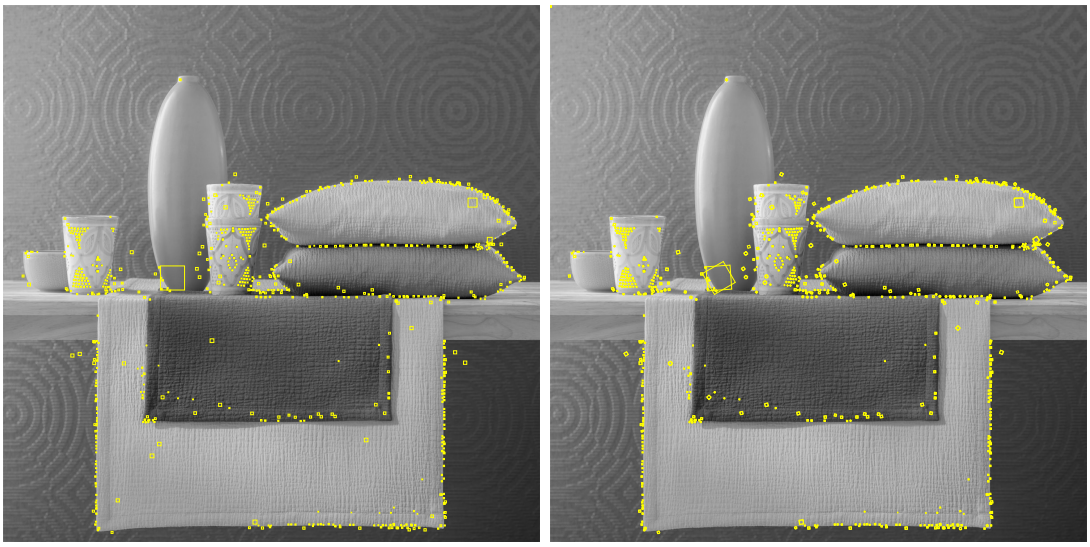

Abbildung 3: Keypoint-Orientierung. Links ohne, rechts mit Orientierung.

Literatur

[Lowe04] David G. Lowe: *Distinctive Image Features from Scale-Invariant Keypoints.* International Journal of Computer Vision 60(2): pp 91-110 (2004)