

Übungsaufgaben III, SBV1

Lukas Fiel, Lisa Panholzer

December 5, 2018

3 Übungsaufgaben III

3.1 Resampling und Bildüberlagerung

a) Zerteilen eines Bildes

Zur vertikalen Teilung eines Bildes wurde ein simpler Filter *ChopImgInHalf* in *ImageJ* implementiert. Dieser definiert zuerst eine ROI (region of interest) welche die erste Hälfte des Bildes beinhaltet. Mittels *ImageJUtility.chopImage* kann dieser Bereich aus dem Ursprungsbild herausgeschnitten und angezeigt werden. Die Berechnung der 2ten Hälfte des Bildes unterscheidet sich lediglich durch die linke obere Koordinate des interessanten Bereichs (roi).







gegebenes Testbild	invertierter Bildausschnitt1	invertierter Bildausschnitt2
		
		

Table 1: Testfälle: Zerteilung eines Bildes

b) Transformation mittels Nearest Neighbor und Bilinearer Interpolation

c) automatische Registrierung

Es wurde ein Filter in *ImageJ* implementiert, der zur automatischen Registrierung von Bildinhalten herangezogen werden soll. Dabei wurde von den gegebenen Testbildern ausgegangen.

Da diese mit einer Bildtiefe von *8bit* nur Werte von 0 (schwarz) bis 255 (weiß) aufweisen, kann mittels SSE einfach ein Algorithmus geschrieben werden, der die Bilder voneinander subtrahiert und die Pixelwerte des Resultatbildes als Fitness heranzieht und aufsummiert. Der Hintergrund der gegebenen Bilder ist dabei meist weiß (255). Bei einer Verschiebung und anschließender Subtraktion entstehen aus diesem Grund aber schwarze Fragmente am Rand. Dieser Umstand kann leicht eliminiert werden, indem das Ursprungsbild zu Beginn invertiert wird. So ist der Hintergrund schwarz (0). Kanten werden dementsprechend weiß (255) dargestellt.

Das invertierte Bild wird anschließend, wie in Punkt a) beschrieben, zerteilt und die Einzelbilder dargestellt.

Die eigentliche Registrierung verschiebt nun Bild1 in x und y Richtung und rotiert dieses auch um jeweils ein Inkrement. Jedes dieser transformierten Bilder wird nun von Bild2 abgezogen und erneut ein Fitnesswert berechnet. Es ist davon auszugehen, dass ein schwarzer Hintergrund (0) abgezogen von einem schwarzen Hintergrund (0) wiederum 0 ergibt. Werden allerdings weiße Pixel von schwarzem Hintergrund abgezogen, oder schwarzer Hintergrund von weißen Linien abgezogen, so erhält man Werte abweichend von 0. Auch Negativwerte sind so denkbar, weshalb diese Differenzwerte zum Quadrat genommen werden. Hierdurch sind Differenzwerte immer positiv.

Wird Bild1 irgendwann genau auf die Position geschoben an der sich Bild2 befindet so subtrahieren sich die weißen Linien im Idealfall zu 0. So kann ein eindeutiger Fitnesswert errechnet werden, der sein Optimum bei 0 findet.

Aus Ressourcengründen werden all die beschriebenen Berechnungen/Verschiebungen mit dem NearesNeighbor Algorithmus berechnet. Ist das Optimum gefunden wird anschließend noch einmal die Transformation mit Bilinearer Interpolation berechnet und von Bild2 subtrahiert. Das Resultatbild wird zum Schluss für den User sichtbar dargestellt um den Erfolg des Filters zu veranschaulichen.







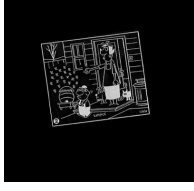
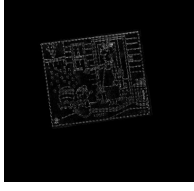

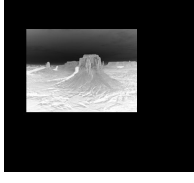
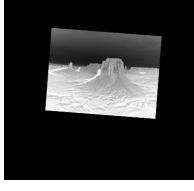
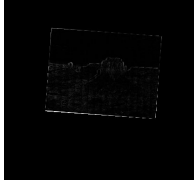
gegebenes Testbild	invertierter Bildausschnitt1	invertierter Bildausschnitt2	resultierendes Differenzbild
			
			
			

Table 2: Testfälle: automatische Registrierung

d) **Optimierungs-Strategie: Gradientenabstieg (gradient descent)
oder Evolutionsstrategie (evolution strategy)**

skipped

e) **Mutual Information Metrik**

skipped

f) **Registrierung mittels DistanceMap**

skipped?!

@Lisa: hier könnte dein Text stehen ;) Im Anschluss findest du noch Code-Beispiele für das Einfügen von Bildern und Tabellen und und und ich habs dir drinnen gelassen, nachdem ich diese selbst verwendet hab ;) bitte löschen wenn du sie nicht mehr brauchst.

Figure 1: Resampling anhand bilinearer Interpolation und Skalierung um Faktor 2.0