Filterung im Ortsbereich

Aufgabenstellung und Randbedingungen:

In drei Teilversuchen sind folgende Filter zu entwickeln:

a) Medianoperator,
b) Sobel-Operator,
c) Binarisierung nach Otsu.

Die korrekte Funktionsweise der Filter ist mit Testbildern zu überprüfen und zu dokumentieren. Der Code ist hinreichend zu kommentieren.

Programmiert wird in Python auf der Basis von Jupyter-Notebooks.

Für alle Teilversuche ist eine Python-Funktion zu schreiben, bei der das/die Ergebnisbild(er) durch Iteration über die Bildpunkte berechnet werden (zwei geschachtelten for-Schleifen). Da Python (ebenso wie Matlab) eine interpretierende Sprache ist, ist dieses Vorgehen sehr (!) ineffizient.

Da es in den Praktikumsversuchen um das Verständnis der Algorithmen geht, wählen wir trotzdem diesen eigentlich ineffizienten Weg, d.h. auf der Basis von zwei geschachtelten for-Schleifen über die Bildpunkte.

Anmerkung zur Effizienz von Python-Programmen:

Sehr effizient ist hingegen (besonders unter Python und Matlab), wenn die durchzuführenden Anwendungen auf der Basis von Vektor- und Matrixoperationen arbeiten (z.B. mit der Python-Bibliothek "numpy"). Das ist auch mit maschinennahem Code kaum zu toppen.

Für Projekte, die eine sehr effiziente Verarbeitung großer Bilddatenmengen erfordern (z.B. Machine-Learning-Projekte), stehen unter Python sehr leistungsfähige Bildverarbeitungsbibliotheken zur Verfügung (skimage, sklearn, opencv, usw.). Diese beinhalten umfangreiche Bildverarbeitungsmethoden, die wir im Praktikum aber eher nicht verwenden.

Allgemeine Implementierungshinweise:

Pro Teilversuch ist ein Jupyter-Notebook zu entwickeln, d.h.
Median.ipynb, Sobel.ipynb, Otsu.ipynb.

Ein Python-Beispielprogramm (*Bildverarbeitungsbeispiel.ipynb*) ist gegeben und zeigt am Beispiel eines Schärfungsoperators, wie die Teilversuche aufgebaut sein können. Die Funktion MySharp(..) ist dann zu ersetzen durch die Funktionen **`MyMedian(..)`** und **`MySobel(..)`, `MyOtsu(..)`**.

Der Code drumherum ist an die jeweilige Aufgabe anzupassen.

Wichtig: Es ist auf korrekte Randbehandlung (besonders beim Median) zu achten, d.h. es darf nicht über den Bildrand hinaus gelesen oder geschrieben werden.

Filterung im Ortsbereich

Hinweise zum Median-Versuch :

Zweck:
Berechnet aus dem Quellbild das Median-gefilterte Bild. Die Größe der Medianmaske soll für die beiden Koordinatenrichtungen (x und y) unabhängig vorgebbar sein. Aus Symmetriegründen sollen nur ungerade Maskengrößen erlaubt sein, z.B. 3x3, 5x17, 51x51. Fehleingaben sollen automatisch korrigiert werden (z.B. 2x16 --> 3x17).

Implementierung:

```
def MyMedian(Quellbild, Maskengröße_x, Maskengröße_y)
...
return Medianbild
```

Testbild : *Shaft.bmp*

Hinweis zur Funktionsweise:
Den Medianwert einer Menge von Grauwerten erhält man z.B. durch folgende Schritte:

a) Sortieren der Maskengrauwerte nach ihrer Größe,
b) Auswahl des in der Mitte der Sortierung stehenden Grauwertes.

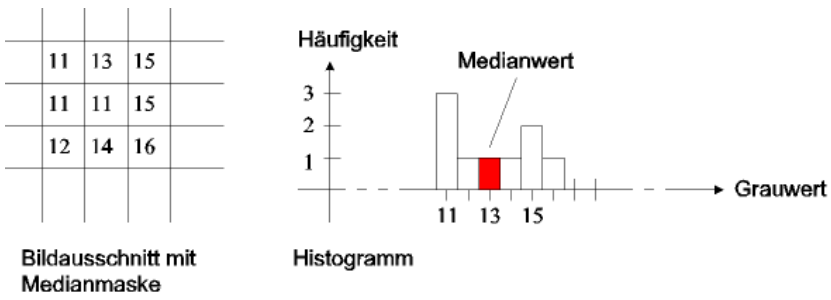
Diese Implementierungsvariante ist jedoch für große Masken aufgrund der Sortierung ineffizient.

Besser ist folgendes Vorgehen (s. nachfolgendes Bild):

a) Für die Grauwerte der Maskenpixel wird das Histogramm berechnet,
b) im Maskenhistogramm wird dann der Medianwert bestimmt.

Anm.: Der Median ist derjenige Grauwert, wo die akkumulierten Histogrammwerte gerade den Wert (Maskengröße_x * Maskengröße_y + 1) / 2 haben.

Die zu entwickelnde Funktion soll nach diesem Histogramm-basierten Verfahren arbeiten.



Für die Ablaufgeschwindigkeit ist es von Vorteil, wenn bei der Histogrammberechnung der Minimalwert der betrachteten Bildumgebung ermittelt wird. Bei der Histogrammsuche kann dann die Suche beim Minimalwert beginnen.

Filterung im Ortsbereich

Hinweise zum Sobel-Versuch :

Zweck:
Berechnet aus dem Quellbild das Gradienten- und Richtungsbild mit dem Sobel-Operator.

Implementierung:

```
def MySobel(Quellbild)
...
return Gradientenbild, Richtungsbild
```

Testbilder : *Shaft.bmp* und
Circle.bmp (besonders geeignet für die Kontrolle des Richtungsbildes).

Hinweis zur Arbeitsweise:
Gradientenbetrag und -richtung eines Bildpunktes erhält man durch

a) Berechnung der Richtungsableitungen G_x und G_y mit den Faltungsmasken:

-1	0	1
-2	0	2
-1	0	1

-1	-2	-1
0	0	0
1	2	1

b) Berechnung des Gradientenbetrages mit: $G = \sqrt{G_x^2 + G_y^2}$

Das Ergebnis ist auf den Zahlenbereich 0..255 zu normieren.

c) Berechnung der Gradientenrichtung mit: $\alpha = \arctan\left(\frac{G_y}{G_x}\right)$
- arctan2(Gy, Gx) verwenden
- Winkel auf (0…360°) normieren.

Filterung im Ortsbereich

Hinweise zum Otsu-Versuch :

Zweck:
Berechnet aus dem Quellbild das automatisch binarisierte Bild (nach der Otsu-Methode).

Implementierung:

```
def MyOtsu(Quellbild)
...
return Binärbild
```

Testbild : *coins.bmp*

Hinweis zur Arbeitsweise:
Siehe Vorlesung ..

Abnahme:

Abgenommen werden die (gut kommentierten) Jupyter-Notebooks der entwickelten Filter, d.h. *Median.ipynb, Sobel.ipynb, Otsu.ipynb*.