Rheinisch Westfälische Technische Hochschule

Institut für Geometrie und Praktische Mathematik

Numerisches Rechnen — WS 2015 / 2016

Prof. Dr. Martin Grepl — Dipl.-Math. Jens Berger — M.Sc. Robert O'Connor

Weihnachtsübung

Abgabe bis 16.00 Uhr am 19.01.2016 in den Kasten vor Raum 102, Hauptgebäude.

1. Einleitung

In dieser Übung geht es um die Kantendetektion in Bildern und die Rekonstruktion eines Bewegungsfeldes aus einer Sequenz aufeinanderfolgender Bilder. Der Einfachheit halber machen wir im folgenden einige Annahmen: Wir beginnen mit der Annahme, dass wir nur graustufige Bilder betrachten. Dann ist das Bild vollständig dadurch charakterisiert, dass man für jeden Pixel einen Wert in $[0, 1]$ hat, der die Helligkeit angibt. Weiterhin machen wir, zumindest für die Herleitung der verschiedenen Methoden, die Kontinuumsannahme. Dies bedeutet, dass für jeden Punkt (x, y, t) in Raum und Zeit die Helligkeit des Bildes als hinreichend glatte Funktion $I : \mathbb{R} \times \mathbb{R} \times \mathbb{R} \to [0, 1]$ $((x, y, t) \to I(x, y, t))$ gegeben ist. Für ein einzelnes Bild haben wir dann $I : \mathbb{R} \times \mathbb{R} \to [0, 1]$ $((x, y) \to I(x, y))$. Dies ist natürlich nur eine Arbeitshypothese, da Bilder im wirklichen Leben digital sind, d.h. dass die Helligkeit nur an diskreten Punkten zur Verfügung steht und die möglichen Werte ebenfalls diskret sind.

2. Finite Differenzen

Im folgenden werden wir die partiellen Ableitungen von I nach x, y und (für die Bildsequenz) nach t benötigen. Da I nicht als analytischer Ausdruck sondern diskret gegeben ist, werden wir die Ableitungen mit finiten Differenzen approximieren. Wir benutzen hier die folgende Notation $I^n_{i,j} = I(x_j, y_i, t^n)$, wobei $x_j := j\Delta x$, $y_i := i\Delta x$, $t^n := n\Delta t$ für Schrittweiten Δx, Δy und Δt größer null. Diese sind in der Praxis durch die Auflösung des Bildes und die Bildfrequenz gegeben. Der Einfachheit halber setzen wir $\Delta x = \Delta y = \Delta t = 1$. Beachten Sie, dass i zur $y-$Komponente und j zur $x-$Komponente gehört. Dies macht man oft in der Bildbearbeitung, wo die Achsen gerne vertauscht werden.

Die Idee der Finiten Differenzen ist es, partielle Ableitungen von I durch den Differenzenquotienten zu approximieren. Ableitungen erster Ordnung können durch den Unterschied in den Werten einer Funktion in zwei Punkten approximiert werden. Für eine Funktion $f : \mathbb{R} \to \mathbb{R}$ und $h > 0$ klein genug haben wir

$$\frac{\partial f(z)}{\partial z} \approx \frac{f(z+h) - f(z)}{h}. \tag{1}$$

Für Helligkeitsableitungen in einem Bild werden wir den Durchschnitt in der anderen Raumdimension berechnen, d.h. wir benutzen zwei Pixel in y-Richtung um die partielle Ableitung in x-Richtung zu berechnen. Wenn wir eine Sequenz von Bildern betrachten, werden wir den Mittelwert der partiellen Ableitungen im Ort für zwei aufeinanderfolgende Bilder benutzen. Für die Ableitung nach der Zeit werden wir den Mittelwert in den beiden Raumdimensionen benutzen. Dies macht die Berechnung der Ableitungen etwas robuster. Die Differenzenquotienten, die wir benutzen werden, sind in Tabelle 1 gegeben. Wegen unserer Annahmen von oben gilt $h = 1$.

	Ableitung	Approximation
Einzelnes Bild	$\frac{\partial I}{\partial x}(j,i)$	$\frac{1}{2}\left(I_{i,j+1} - I_{i,j} + I_{i+1,j+1} - I_{i+1,j}\right)$
	$\frac{\partial I}{\partial y}(j,i)$	$\frac{1}{2}\left(I_{i+1,j} - I_{i,j} + I_{i+1,j+1} - I_{i,j+1}\right)$
Sequenz	$\frac{\partial I^k}{\partial x}(j,i)$	$\frac{1}{4}\left(I^k_{i,j+1} - I^k_{i,j} + I^k_{i+1,j+1} - I^k_{i+1,j} + I^{k+1}_{i,j+1} - I^{k+1}_{i,j} + I^{k+1}_{i+1,j+1} - I^{k+1}_{i+1,j}\right)$
	$\frac{\partial I^k}{\partial y}(j,i)$	$\frac{1}{4}\left(I^k_{i+1,j} - I^k_{i,j} + I^k_{i+1,j+1} - I^k_{i,j+1} + I^{k+1}_{i+1,j} - I^{k+1}_{i,j} + I^{k+1}_{i+1,j+1} - I^{k+1}_{i,j+1}\right)$
	$\frac{\partial I^k}{\partial t}(j,i)$	$\frac{1}{4}\left(I^{k+1}_{i+1,j} - I^k_{i+1,j} + I^{k+1}_{i+1,j+1} - I^k_{i+1,j+1} + I^{k+1}_{i,j+1} - I^k_{i,j+1} + I^{k+1}_{i,j} - I^k_{i,j}\right)$

Tabelle 1: Approximationen für Ableitungen erster Ordnung

Bemerkung: Die Ableitungen in der Tabelle 1 sind in Wahrheit bessere Approximationen für die Werte mit $t = k + 1/2$, $x = j + 1/2$ und $y = i + 1/2$ (Stichwort zentrale Finite Differenzen). Für die Notation ist es aber einfacher, diese als Approximationen an den Stellen $t = k$, $x = j$ und $y = i$ zu betrachten. Auf diese Weise passen die Werte auch direkt in eine Matrix.

Eine besondere Größe, die wir benötigen, ist der Gradient:

$$\nabla I(j,i) = \left(\frac{\partial I}{\partial x}(j,i), \frac{\partial I}{\partial y}(j,i)\right)^T. \tag{2}$$

Der Gradient wird approximiert, indem man die jeweiligen Differenzenquotienten aus Tabelle 1 einsetzt.

Für Ableitungen zweiter Ordnung benötigt man Werte an mindestens drei Punkten.

$$\frac{\partial^2 f(z)}{\partial z^2} \approx \frac{f(z+h) - 2f(z) + f(z-h)}{h^2} \tag{3}$$

Insbesondere werden wir den Laplace-Operator Δ benötigen.

$$\Delta = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} \tag{4}$$

Der Laplace-Operator ist einfach die Summe der Ableitungen zweiter Ordnung.

Der Einfachheit halber werden wir bei Ableitungen zweiter Ordnung keine Durchschnitte berechnen. In Tabelle 2 sind die Differenzenquotienten zusammengefasst, die wir benutzen werden.

Ableitung	Approximation
$\frac{\partial^2 I}{\partial x^2}(j,i)$	$I_{i,j+1} - 2I_{i,j} + I_{i,j-1}$
$\frac{\partial^2 I}{\partial y^2}(j,i)$	$I_{i+1,j} - 2I_{i,j} + I_{i-1,j}$
$\Delta I(j,i)$	$I_{i+1,j} + I_{i-1,j} + I_{i,j+1} + I_{i,j-1} - 4I_{i,j}$

Tabelle 2: Approximationen für Ableitungen zweiter Ordnung

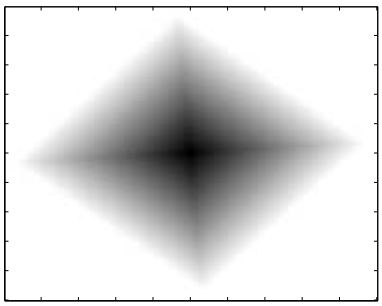
2.1. Randbedingungen

Um die Approximationen in den Tabellen 1 und 2 auszuwerten, benötigt man am Rand des Bildes zusätzliche Werte. Häufig geht man hier von einer Neumann-Randbedingungen aus: Wir haben Werte für $I_{j,i}$ für alle $i = 1, 2, \ldots, n$ und $j = 1, 2, \ldots, m$ und nehmen an, dass sich die Helligkeit am Rand in der Richtung, die zum Rand senkrecht ist, nicht ändert. Dann gilt:

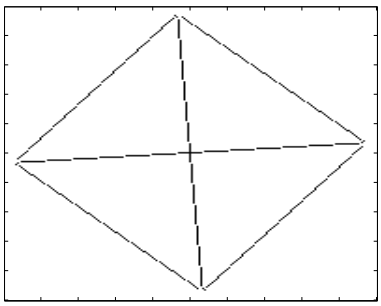
$$I_{0,j} := I_{1,j}, \qquad \forall j = 1, 2, \ldots, m, \tag{5a}$$
$$I_{n+1,j} := I_{n,j}, \qquad \forall j = 1, 2, \ldots, m, \tag{5b}$$
$$I_{i,0} := I_{i,1}, \qquad \forall i = 1, 2, \ldots, n, \tag{5c}$$
$$I_{j,m+1} := I_{j,m}, \qquad \forall i = 1, 2, \ldots, n. \tag{5d}$$

3. Kantendetektion

Mit Hilfe der Kantendetektion versucht man, die Kanten in einem Bild zu erkennen. Diese Kanten sind oft Sprünge entweder in der Helligkeit oder im Gradienten der Helligkeit. Abbildung 1 zeigt ein Beispiel, in dem man den Laplace-Operator benutzen kann, um Sprünge im Gradienten der Helligkeit zu finden.



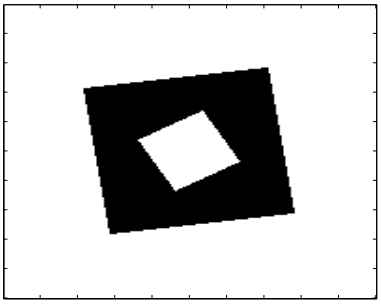
Bild



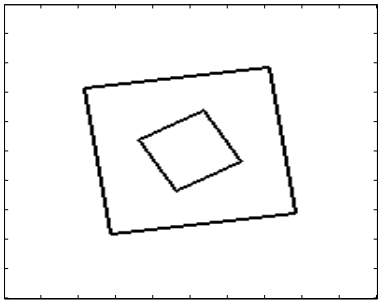
Kanten

Abbildung 1: In diesem Bild gibt es keinen Sprung in der Helligkeit. Deshalb helfen Gradienten nicht bei der Suche nach Kanten. Man kann stattdessen den Laplace-Operator benutzen.

Wir werden im folgenden den einfacheren Fall betrachten, d.h. Sprünge in der Helligkeit. In solchen Fällen kann man den Gradienten benutzen, um die Kanten zu erkennen. Genauer gesagt berechnet



Bild



Kanten

Abbildung 2: Kantendetektion mit $(\nabla I)^T(\nabla I)$.

man überall $(\nabla I)^T(\nabla I)$ mit Hilfe der oben erwähnten Differenzenquotienten. Man bekommt dann eine Funktion, die an den Kanten im Bild sehr große Werte hat.

Aufgabe 1: (Kantendetektion) [10 Punkte]
Versuchen Sie die Kanten in den Bildern *drei.bmp* und *rau.bmp* hervorzuheben. Berechnen Sie $(\nabla I)^T(\nabla I)$ und ΔI für alle $i = 1, 2, \ldots, n$ und $j = 1, 2, \ldots, m$ wobei n und m die Anzahl der Spalten und Zeilen des Bilds sind. Erstellen Sie dann Bilder mit diesen Werten. Drucken Sie Ihre vier Bilder aus, und geben Sie sowohl die Bilder als auch Ihren Code ab. Bei welchen Bildern hat die Kantendetektion funktioniert? Wieso hat sie bei den anderen nicht funktioniert? Sind die Ergebnisse mit $(\nabla I)^T(\nabla I)$ anders als die Ergebnisse mit ΔI? Wenn sie anders sind, wieso?
Hinweis: Die Bilder müssen nicht unbedingt als bmp-Dateien gespeichert werden.
Bemerkung: Wenn die Kantendetektion gut funktioniert hat, sollte es möglich sein, die Kanten über einen Schwellenwert genau zu bestimmen. Das ist hier aber nicht gefordert!

Aufgabe 2: (Kantendetektion) [4 Punkte]
Benutzen Sie nun Schwellenwertbinarisierungen und die Methode, die wir gesehen haben, um aus *drei.bmp* zwei binäre Bilder zu erzeugen. Das erste soll ausschließlich die zwei kleinen Kreise zeigen und ansonsten weiß sein. Das zweite soll nur den Rand des großen Kreises zeigen und ansonsten weiß sein.
Hinweis: Im Zentrum des Bildes wird es vielleicht Schwierigkeiten geben. Wenn dort einige schwarze Punkte auftauchen, ist das kein Problem.

4. Entrauschung

Die meisten realen Bilder sind verrauscht, z.B. SAR Bilder oder MRT Bilder in der Medizin. Ziel des Bildentrauschens ist es, hochfrequente Störungen aus einem Bild zu entfernen. Dazu existieren viele verschiedene Methoden. Wir werden eine Methode betrachten, die von der Wärmeleitungsgleichung inspiriert ist. Sei $u(x, y, t)$ die Temperatur an einer Stelle (x, y) zur Zeit t. Die Wärmeleitungsgleichung lautet dann

$$\frac{\partial}{\partial t}u(x, y, t) = a\Delta u(x, y, t), \tag{6}$$

wobei a die Diffusionskonstante ist. Wir werden die Wärmeleitungsgleichung simulieren und dabei ein Bild als Anfangswert benutzen. Die Zeit dient hier als künstlicher Parameter, mit Hilfe dessen das Entrauschen gesteuert werden kann: Mit zunehmender Zeit wird mehr Rauschen entfernt und das Bild „glatter“ werden. Das Ziel ist es, das Rauschen zu reduzieren, ohne dabei jedoch die wesentlichen Eigenschaften des Bildes zu verlieren. Eine wichtige Annahme dabei ist, dass das Rauschen eine höhere Frequenz hat als andere Komponenten des Bildes. Trotzdem muss man bedenken, dass das

Bild komplett „glatt“ – d.h. ein konstanter Helligkeitswert – und damit nutzlos wird, wenn man die Wärmeleitungsgleichung bis zur stationären Lösung laufen lässt.

Für das zu entrauschende Bild I werden wir die folgende Diskretisierung von (6) benutzen

$$\begin{aligned} I_{i,j}^{\Delta,k+1} &= I_{i,j}^{\Delta,k} + \Delta t \cdot a \hat{\Delta} I_{i,j}^{\Delta,k}, \\ I_{i,j}^{\Delta,0} &= I_{i,j}, \end{aligned} \tag{7}$$

wobei $\hat{\Delta} I$ die in Tabelle 2 gegebene diskrete Approximation von ΔI ist. Der Einfachheit halber setzen wir $\Delta t = 1$. Es genügt dann, passende Parameter a und K zu finden, sodass $I_{i,j}^{\Delta,K}$ das entrauschte Bild ist.

Aufgabe 3: (Kantendetektion mit Entrauschung) [10 Punkte]
Schreiben Sie ein Programm, um das oben beschriebene Entrauschungsverfahren durchzuführen. Benutzen Sie dieses Programm um das Bild *rau.bmp* mit $a = .006$ und $K = 250$ zu entrauschen. Versuchen Sie dann die Kanten zu detektieren. Benutzen Sie dabei $(\nabla I)^T(\nabla I)$ und ΔI (vgl. Aufgabe 1). Welcher Operator funktioniert am besten bei welcher Art von Kanten? Wieso?

5. Echte Bilder

Bis jetzt haben wir nur sehr einfache Bilder betrachtet. Die gleichen Werkzeuge können aber auch auf echte Bilder angewendet werden. Das Bild *camera.bmp* ist ein bekanntes Bild, das auch in Matlab zur Verfügung steht.

Aufgabe 4: (Kantendetektion) [6 Punkte]
Schreiben Sie ein Programm, um die Kanten im Bild *camera.bmp* zu bestimmen. Benutzen sie dabei eine Schwellwertbinärisierung. Die Kanten sollten in schwarz sein und alles andere sollte weiss sein. Kleine Variationen im Himmel und im Rasen sind nicht als Kanten zu betrachten, aber die Silhouette des Mannes und des Stativs sollten deutlich erkennbar sein.

6. Optischer Fluss

In der letzten Aufgabe geht es um die Rekonstruktion des scheinbaren Bewegungsfeldes aus einer Sequenz aufeinanderfolgender Bilder. Eine fundamentale Annahme wird diejenige sein, dass sich die Helligkeit eines Objekts von einem Bild zum nächsten nicht ändert. Das Objekt (und die zugehörigen Pixel) können sich natürlich „bewegen“, dabei verändert sich jedoch nicht die entsprechende Helligkeit des betrachteten Objekts. Mathematisch ausgedrückt bedeutet dies, dass für Pixel, die sich auf einer Trajektorie $(\widetilde{x}(t), \widetilde{y}(t))$ bewegen, gilt:

$$\frac{d}{dt} I(\widetilde{x}(t), \widetilde{y}(t), t) = 0. \tag{8}$$

Gleichung (8) wird die für unseren Algorithmus grundlegende Gleichung sein. In der englischen Literatur nennt man diese Gleichung die *Image Brightness Constancy Equation*. Mittels Kettenregel umgeformt ergibt sich

$$\frac{\partial}{\partial x} I(\widetilde{x}(t), \widetilde{y}(t), t) \frac{d\widetilde{x}}{dt} + \frac{\partial}{\partial y} I(\widetilde{x}(t), \widetilde{y}(t), t) \frac{d\widetilde{y}}{dt} + \frac{\partial}{\partial t} I(\widetilde{x}(t), \widetilde{y}(t), t) = 0. \tag{9}$$

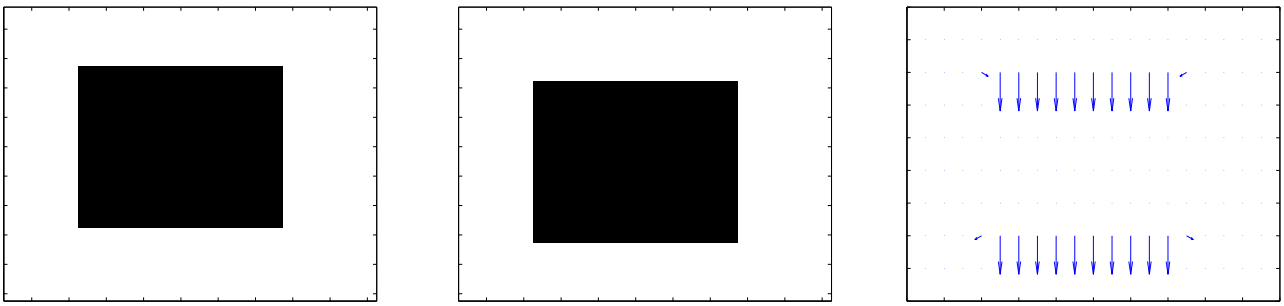
Wir definieren den sogenannten *optischen Fluss* (u, v) nun als

$$(u, v) := \left(\frac{d\widetilde{x}}{dt}, \frac{d\widetilde{y}}{dt} \right). \tag{10}$$

Wir können (9) dann als

$$I_x u + I_y v + I_t = 0 \tag{11}$$

schreiben, wobei hier alle Variablen von x, y und t abhängig sind, und I_x, I_y und I_t die partielle Ableitung von I nach x, y und t sind. Abbildung 3 zeigt einen lokal berechneten optischen Fluss für ein Quadrat, das im Bild nach unten wandert.



1. Bild 2. Bild Optischer Fluss

Abbildung 3: Eine Sequenz von 2 Bildern, in denen sich das Viereck nach unten bewegt, sowie ein lokal berechneter optischer Fluss

In Wahrheit hat (11) meistens keine eindeutige Lösung (siehe auch das sogenannte Aperture Problem, d.h. die Komponente des optischen Flusses rechtwinklig zum Gradieten ∇I ist immer unbekannt). In Regionen, in denen die Helligkeit konstant ist, kann man außerdem nicht sagen, ob sich die Objekte bewegen. Mit der Annahme, dass alle Bewegungen minimal sind, bekommt man einen optischen Fluss, wie in der Abbildung 3 gezeigt.

7. Bestimmung des optischen Flusses (u, v)

Ausgehend von (9) wollen wir u und v approximativ bestimmen. Dazu ersetzen wir in jedem Gitterpunkt die partiellen Ableitungen durch ihre Approximationsformeln aus Sektion 2. Man beachte, dass sich die Einträge mit $k+1$ auf das zweite Bild *bewegung2.bmp* beziehen. Die Idee des Algorithmus ist nun die folgende:

a) Berechnen Sie I_x, I_y und I_t.

b) Betrachten Sie für jeden Pixel (i_0, j_0) ein kleines Quadrat aus Pixeln mit Mittelpunkt (i_0, j_0), einen sogenannten *Patch* $\mathcal{P}_{i_0,j_0}$. Benutzen Sie eine Dimensionierung von 5x5. An den Grenzen des Gebiets können Sie den *Patch* soviel kleiner machen wie notwendig.

c) Suchen Sie nun einen konstanten Vektor $(u^\star, v^\star) \in \mathbb{R}^2$, so dass

$$\sum_{i,j \in \mathcal{P}_{i_0,j_0}} |I_x(i,j)u^\star + I_y(i,j)v^\star + I_t(i,j)|^2 \to \min_{u,v \in \mathbb{R}} \sum_{i,j \in \mathcal{P}_{i_0,j_0}} |I_x(i,j)u + I_y(i,j)v + I_t(i,j)|^2. \tag{12}$$

Dies führt für jeden Pixel auf ein lineares Ausgleichsproblem der Form

$$\|Ax - b\|_2 \to \min_{x \in \mathbb{R}^2}. \tag{13}$$

Bei 5x5 Patches bekommt man ein lineares Ausgleichsproblem mit Dimensionen $m = 25$ und $n = 2$, das heißt die Normalengleichungen stellen ein lineares Gleichungssystem der Größe 2x2

dar. Sie können das lineare Ausgleichsproblem also direkt lösen. Beachten Sie allerdings, dass die zugehörige Matrix A auch singulär, d.h. $\text{Rang}(A) < 2$, sein kann (dies tritt häufig auf, nämlich dort, wo sich die Helligkeit nicht ändert). Benutzen sie deshalb die Pseudoinverse von $A^T A$, anstatt zu versuchen die Normalgleichung zu lösen (vgl. Kapitel 4.7 im Buch).

Aufgabe 5: (Optischer Fluss) [10 Punkte]
Berechnen Sie für das Bildpaar *bewegung*1.*bmp* und *bewegung*2.*bmp* den optischen Fluss. Ergibt Ihr berechneter optischer Fluss Sinn?

8. Abgabe

Drucken sie Ihren Code und die Bilder aus, um sie abzugeben. Der Code soll verständlich kommentiert sein.

A. Matlab

Matlab verfügt über viele nützliche Werkzeuge, die bei diesem Übungsblatt verwendet werden können. Um bmp-Dateien zu öffnen, gibt es die Funktion *imread*. Die Rückgaben von *imread* sind Matrizen mit ganzzahligen Einträgen. Bevor man mit diesen Matrizen arbeiten kann, muss man die Werte mit Hilfe von *double* in Gleitkommazahlen mit doppelter Genauigkeit konvertieren. Man könnte die Einträge auch skalieren, wenn man Werte in $[0, 1]$ haben will. Um Bilder zu plotten, stehen *imshow* und *imagesc* zur Verfügung. Wenn man *imagesc* benutzt, bekommt man farbige Bilder. Um die Bilder mit Graustufen anzuzeigen, benutzt man den Befehl *colormap(gray)*.

Um optische Flüsse zu plotten, kann man *quiver* benutzen. Eine Schwierigkeit im Bezug auf *quiver* ist, dass die Pfeile oft zu klein werden. Man kann einen optischen Fluss mit folgendem Befehl auf einem groben Gitter plotten: *quiver(u(1:10:end,1:10:end),v(1:10:end,1:10:end))*.