Optischer Fluss und Echtzeit-Videobearbeitung

Wolfgang Konen
Institut für Informatik, FH Köln

17.10.2006

Abstract. Dieser kurze Technical Report enthält einige Anmerkungen zum Optischen Fluss und Erläuterungen zum Algorithmus in [Kourogi99], der den Optischen Fluss zur Erstellung von Übersichtsbildern (Panoramas) aus einer Videosequenz nutzt. Ziel ist es, die recht knapp gehaltene Arbeit [Kourogi99] durch einige Vertiefungen besser verständlich zu machen.
Es werden Anmerkungen zu einer exemplarischen MATLAB-Implementierung des Algorithmus gegeben.

Optischer Fluss

Nehmen wir an, dass wir uns in einer statischen Welt befinden.

Bewegt sich ein Beobachter (eine Kamera) durch diese statische Welt, so entsteht auf seiner Retina (auf dem Kameratarget) dennoch Bewegung, der sog. "Optische Fluss" (engl. *optical flow*). Unter dem optischen Fluss versteht man das Verschiebungsvektorfeld (engl. *motion field*) $\begin{pmatrix} u(x,y) \\ v(x,y) \end{pmatrix}$, das für jeden Punkt der Welt angibt, welche Verschiebung auf der Bildebene er in einem Zeitintervall Δt erfährt. Abbildung 1 zwei Beispiele für optischen Fluss: Links der optische Fluss, wenn man sich nach hinten bewegt, denn dann wandern alle Punkte der statischen Welt einwärts. Rechts der optische Fluss, wenn man den Kopf nach rechts dreht, denn dann wandert alle Punkte der statischen Welt nach links.

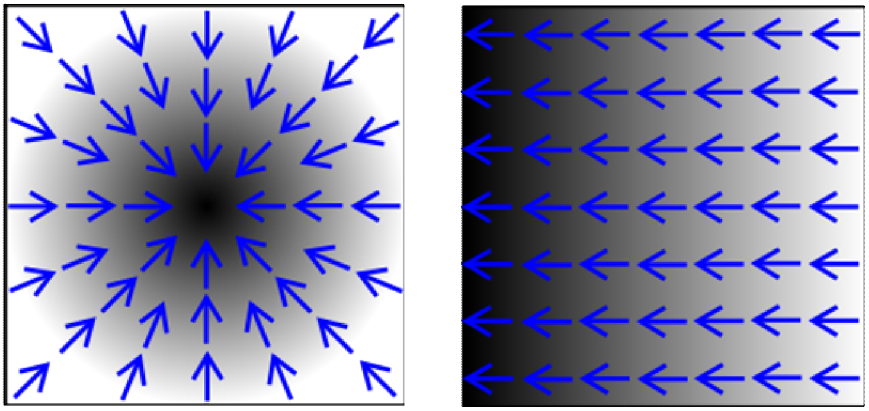

Abbildung 1: Zwei Beispiele für optischen Fluss. Dargestellt sind einige Vektoren (Pfeile) aus einem kontinuierlichen Verschiebungsvektorfeld.

Es sei $\begin{pmatrix} u \\ v \end{pmatrix}$ der wahre Verschiebungsvektor für Pixel (x,y). Eine übliche Annahme der Computer Vision ist nun, das sich die Lichtstärke, die von einem Punkt der statischen Welt ausgeht, im Zeitintervall Δt nicht ändert. (Wir sehen also von Glühwürmchen, Taschenlampen und Pulsaren für den Moment ab.) Dann können wir schreiben

(1) $$I(x+u, y+v, t) - I(x, y, t-\Delta t) = 0$$

Diese Gleichung bedeutet einfach: Die Lichtstärke, die vorher, also zum Zeitpunkt *t-Δt,* im Bildpunkt *(x,y)* war, befindet sich zum Zeitpunkt *t* im Bildpunkt *(x+u,y+v)*.

Wenn wir annehmen, dass die Intensität I eine stetige Funktion ist – eine Annahme, die in der Realität wegen Sprüngen im Grauwertverlauf leider öfters *nicht* zutrifft – dann können wir Gl. (1) für kleines Δt, mithin auch kleines u und v, in eine Reihe entwickeln (linearisieren) und erhalten

$$I(x+u, y+v, t) - I(x, y, t-\Delta t) = 0$$
$$I(x, y, t) + I_x u + I_y v - \left(I(x, y, t) - I_t \Delta t\right) = 0$$

(2) $$\boxed{I_x u + I_y v + I_t \Delta t = 0}$$

Hierbei sind I_x, I_y und I_t die partiellen Ableitungen von I nach x, y und t. Gl. (2) ist die berühmte, von Horn und Schunck [HornSchunck81] entwickelte Gleichung des optischen Flusses. Wenn wir auf das Zeitinkrement $\Delta t=1$ spezialisieren (von einem Videoframe zum nächsten) und eine andere Schreibweise für die partielle Ableitung verwenden, so erhalten wir als äquivalente Variante eine Form, die sich oft in der Literatur (so auch bei [Kourogi99]) findet:

(2') $$\boxed{\frac{\partial I}{\partial x} u + \frac{\partial I}{\partial y} v + \frac{\partial I}{\partial t} = 0}$$

Die Methode nach Kourogi'99

Ziel: Gegeben seien zwei aufeinander folgende Frames aus einem Video. Wie kann man das Verschiebungsvektorfeld zwischen ihnen schätzen, und das möglichst in Echtzeit?

Es sei wiederum $\begin{pmatrix} u \\ v \end{pmatrix}$ der wahre Verschiebungsvektor für Pixel *(x,y)* bei $\Delta t=1$. D. h. es gelte[1]

$$I(x+u, y+v, t) - I(x, y, t-1) = 0$$

Leider kennen wir den wahren Verschiebungsvektor in der Regel nicht! Was tun? – Die berühmte Gl. (2) hat einen bekannten, entscheidenden Nachteil: Sie ist *allein* und ist so nicht hinreichend dafür, die *zwei* Unbekannten u und v zu bestimmen.

Pseudo Motion

[Kourogi99] geht einen auf den ersten Blick brutal vereinfacht anmutenden Weg:

- Wenn wir u ausrechnen wollen, dann setzen wir in Gl. (2) einfach den v-Term zu Null
- Wenn wir v ausrechnen wollen, dann setzen wir in Gl. (2) einfach den u-Term zu Null

Dies führt auf die sog. **Pseudo Motion**:

(3) $$\begin{pmatrix} u_p \\ v_p \end{pmatrix} = \begin{pmatrix} -I_t / I_x \\ -I_t / I_y \end{pmatrix},$$

Hierbei sind I_x, I_y und I_t die partiellen Ableitungen von I nach x, y und t, die durch ihre diskreten numerischen Vertreter (s.u., Implementierungsaspekte) anzunähern sind. Die Gl. (3) darf natürlich nur dort

[1] In [Kourogi99] wird das Frame zum Zeitpunkt t wird auch als "current" oder I_c, das zum Zeitpunkt t-1 auch als "reference" oder I_r bezeichnet.

benutzt werden, wo I_x bzw. I_y nicht Null ist. Dieser Pseudo Motion Vektor ist zwar oftmals ungenau, aber nimmt man viele Samples so sollte, unter recht allgemeinen Annahmen, die stochastische Größe $u_p - u$ um den Mittelwert 0 streuen. Weil aber bei der Pseudo Motion oft auch 'wirre' Werte vorkommen werden, führt [Kourogi99] folgenden Test durch:

(4) $$\left| I(x+u_p, y+v_p, t) - I(x,y,t-1) \right| < T \,,$$

wobei T eine geeignet zu wählende Grauwertschwelle ist, z.B. $T=5$. Nur Pseudo Motion Vektoren, die diesen Test passieren, werden zur weiteren Analyse herangezogen.

Die Sache mit der Pseudo Motion hat also den Vorteil, dass wir sehr schnell Werte für u und v ausrechnen können. Sie hat den Nachteil, dass sie manchmal funktioniert, aber oft auch nicht funktioniert. Bevor wir weitergehen, machen wir uns erst einmal anschaulich klar, wann es funktionieren kann und wann es schief gehen muss:

Wann ist der Pseudo Motion Vektor genau? – Wenn vom Zeitpunkt t-1 bis t eine **rein lineare** Grauwertrampe am Beobachtungsort (x,y) vorbeiwandert. Nehmen wir als Beispiel eine 1-dimensionale (1D) Bewegung in x, bei der eine Grauwertrampe [130 132 134 136 138] mit einer Geschwindigkeit von –3 Pixel/Frame am Beobachtungsort vorbeiwandert. Zum Zeitpunkt t-1 sei der Grauwert am Beobachtungsort 132, dann ist er also 138 zum Zeitpunkt t am Beobachtungsort. Wir errechnen $I_t = 138–132 = 6$ und $I_x = (134–130)/2 = 2$ und Gl. (1) führt richtigerweise auf $u_p = –6/2 = –3$.

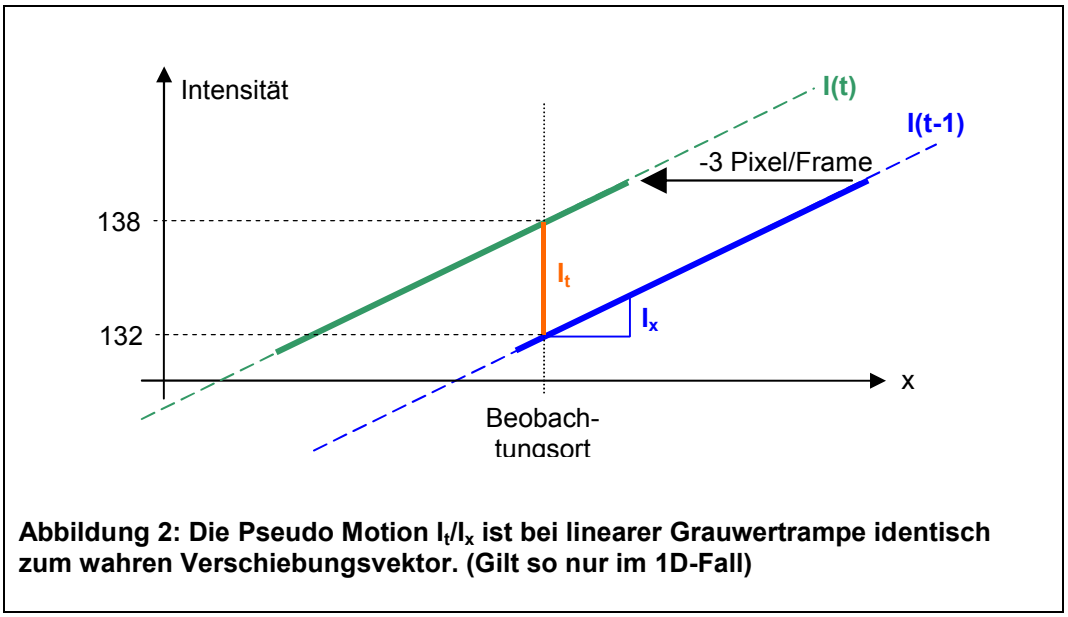

Abbildung 2: Die Pseudo Motion I_t/I_x ist bei linearer Grauwertrampe identisch zum wahren Verschiebungsvektor. (Gilt so nur im 1D-Fall)

Wir sehen umgekehrt daraus auch, wann der Pseudo Motion Vektor falsch sein muss:

- wenn I_x oder I_y Null werden, dann ist der Quotient in (3) ja auch nicht definiert. Wir können auch prinzipiell nichts ausrechnen, weil der lokale Gradient keine Info liefert (in Bereichen ohne Struktur kann man keinen Verschiebungsvektor errechnen)
- wenn der Grauwertbereich, der von t-1 bis t am Beobachtungsort (x,y) vorbeiwandert, eine Sprungstelle (Diskontinuität, engl. *discontinuity*) enthält, was ja bei Bildern häufig der Fall sein kann. Dann hat der zeitliche Verlauf in I_t Änderungen "gesehen", die der lokale Gradient, also die Ableitung nach x, nicht kennen kann.
- Die Wahrscheinlichkeit für Sprungstellen wird natürlich umso größer, je größer die Verschiebungsvektoren sind

Eine Möglichkeit, den störenden Einfluss der Diskontinuitäten zurückzudrängen, besteht in Compensated Motion, s. nächster Abschnitt.

Weitere Faktoren, die zur Ungenauigkeit im Pseudo Motion Vektor beitragen:

- Quantisierung im Grauwertbereich >> Rauschen

- zwar keine unstetigen, aber nicht-lineare Grauwertverläufe (z.B. quadratisch). Der Effekt wird natürlich umso stärker, je größer der Motion Vektor ist. Auch hier kann Compensated Motion helfen.

Implementierungsaspekte:

- WICHTIG: Der Test in Gl. (4) ist mit Sub-Pixel-Genauigkeit durchzuführen, also z.B. bilineare Interpolation für den ersten I-Grauwert verwenden.
- Die diskreten Ableitungen sind:
  $$I_t = I(x,y,t) - I(x,y,t-1)$$
  $$I_x = \big(I(x+1,y,t-1) - I(x-1,y,t-1)\big)/2$$
  $$I_y = \big(I(x,y+1,t-1) - I(x,y-1,t-1)\big)/2$$
  (wobei natürlich auch andere, kürzer- oder längerreichweitige Varianten der Ableitungen denkbar sind)
- Implementierung in <MATLAB>/real_mos/pmotion.m (Step 1+2) und test_pmotion.m für `i-termax=1`:
  - Das Gültigkeitsarray `accept` ist =1 für alle Pixelorte (x,y), für die gilt
    - ($x+u_p$, $y+v_p$) liegt auch noch im gültigen, der bilinearen Interpolation zugänglichen Bereich (`checkInMask`)
    - I_x und I_y sind ungleich Null
    - Der Test nach Gl. (4) ist erfüllt.

Wenn wir eine rein translatorische Bewegung haben, dann haben alle Pixel den gleichen Verschiebungsvektor. Wir können die unvermeidliche Streuung, die die Pseudo Motion hat, dadurch verringern, dass wir über viele Pixel mitteln. Wir berechnen einfach

(5) $$\bar{u}_p = \frac{1}{|A|}\sum_{p\in A} u_p\,, \qquad \bar{v}_p = \frac{1}{|A|}\sum_{p\in A} v_p$$

Für kleine Verschiebungen von ungefähr 1 Pixel wird eine solche Schätzung recht genau, wenn die Menge A der akzeptierten Pixel genügend Elemente enthält (|A| ist die Anzahl der Elemente der Menge A)

Compensated Motion

Wir wollen aber auch größere Verschiebungen als ± 1 Pixel detektieren. Hier stoßen wir natürlich mit der Pseudo Motion schnell an Grenzen, wie die folgende Tabelle zeigt (test_pmotion2.m):

Tabelle 2: Pseudo Motion für größere Translationen. Angaben in Pixeln.

T=5		wahre Translation		Pseudo Motion		
		u	v	$\bar{u}_p$	$\bar{v}_p$	% accept
		-1.5	-2.5	-0.6 ± 4.7	-1.4 ± 5.0	30%
		-5.2	-3.5	-1.5 ± 6.8	-0.8 ± 8.5	21%
		-10.5	7.6	-1.5 ± 10.0	2.2 ± 12.3	14%

Die Pseudo Motion "springt also zu kurz", bzw. wenn man genauer hinschaut, dann wird vor allem die Streuung immer größer. Dies ist auch nicht verwunderlich: Mit größer werdender Translation wird der Grauwertverlauf, der von t-1 bis t durch den Beobachtungspunkt geschoben wird, auch immer größer. Es wächst die Wahrscheinlichkeit, dass darin Diskontinuitäten enthalten sind oder dass die lineare Steigung, die mit dem Gradienten (I_x,I_y) geschätzt wird nicht die durchschnittliche Steigung von t-1 bis t ist.

[Kourogi99] schlägt die Compensated Motion als Lösung vor. Die Idee dahinter ist die folgende: Wenn wir aus irgendwelchen Quellen eine – wenn auch nur grobe – Schätzung $\begin{pmatrix} u_c \\ v_c \end{pmatrix}$ für das Verschiebungsvektorfeld haben, dann können wir diese Verschiebungskomponente schon einmal überall kompensieren. Der verbleibende Rest $\begin{pmatrix} u_r \\ v_r \end{pmatrix}$ ist dann (hoffentlich!) kleiner als das gesamte $\begin{pmatrix} u \\ v \end{pmatrix}$. Wenn wir also in einem 2. Schritt diesen verbleibenden Rest $\begin{pmatrix} u_r \\ v_r \end{pmatrix}$ bestimmen wollen, dann haben wir eine weniger große Strecke zu überwinden.

Wie aber bekommen wir eine (grobe) Schätzung? – Wir nehmen, ähnlich wie beim Warping, an, dass das ganze Verschiebungsfeld aus einer bestimmten Klasse von Transformationen stammt, z.B. "affin" oder "Translation". Aus der Pseudo Motion wird (mit LS-Schätzung affin oder nur translatorisch) eine globale Compensated Motion geschätzt:

(6) **affin:** $\begin{pmatrix} u_c \\ v_c \end{pmatrix} = \begin{pmatrix} a_1 x + a_2 y + a_3 \\ a_4 x + a_5 y + a_6 \end{pmatrix}$ **oder translatorisch:** $\begin{pmatrix} u_c \\ v_c \end{pmatrix} = \begin{pmatrix} \bar{u}_p \\ \bar{v}_p \end{pmatrix}$

Im Frame zum Zeitpunkt t kann nun die geschätzte Bewegung kompensiert werden. Dies führt auf eine andere partielle Ableitung nach der Zeit (I_x und I_y ändern sich nicht, da sie im Frame t-1 berechnet werden) und eine modifizierte Pseudo Motion Berechnung

$$I_t^{(c)} = I(x + u_c, y + v_c, t) - I(x, y, t-1)$$

(7)
$$\begin{pmatrix} u_p \\ v_p \end{pmatrix} = \begin{pmatrix} -I_t^{(c)} / I_x \\ -I_t^{(c)} / I_y \end{pmatrix} + \begin{pmatrix} u_c \\ v_c \end{pmatrix}$$

Das neue (u_p,v_p) wird wieder dem Gültigkeitstest nach Gl. (4) unterworfen. Das Verfahren der wechselnden Berechnung von (u_c,v_c) und (u_p,v_p) wird nun iteriert und führt zu einer verbesserten Schätzung der Motion-Vektoren.

Tabelle 3: Pseudo Motion, iteratives Verfahren mit Compensated Motion. Angaben in Pixeln.

T=5		wahre Translation		Pseudo Motion		
		u	v	$\bar{u}_p$	$\bar{v}_p$	% accept
iter	1	-10.5	7.6	-1.5 ± 10.0	2.2 ± 12.3	14%
	2	-10.5	7.6	-3.1 ± 8.6	3.8 ± 10.3	16%
	5	-10.5	7.6	-7.4 ± 5.3	7.0 ± 6.8	23%
	10	-10.5	7.6	-10.47 ± 1.2	7.61 ± 1.27	55%

(die fehlenden Iterationen wurden gerechnet, nur in der Tabelle ausgelassen)

Man sieht, dass es so auch für große Translationen geht! Recht schnell erzielt man Konvergenz und das mit Subpixelgenauigkeit. Im Laufe der Iterationen erfüllen sehr viel mehr Pixel den Genauigkeitstest als es zu Anfang der Fall war.

Warum funktioniert es mit Compensated Motion besser? – Dies lässt sich an folgendem Bild erklären:

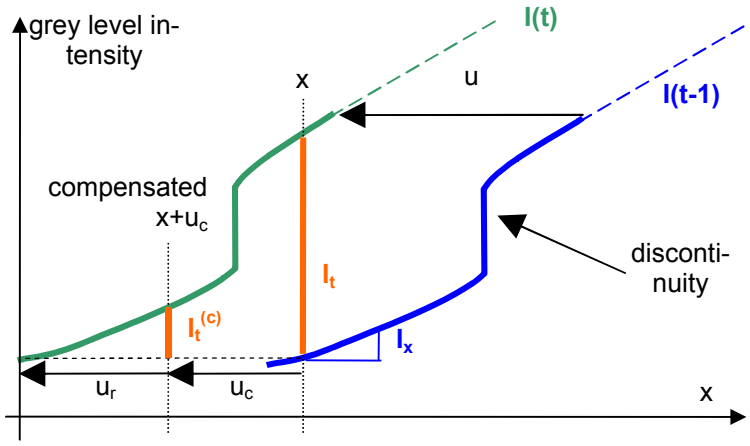

Abbildung 3: Zur Funktionsweise der Compensated Motion

Wir stehen im Beobachtungsort x und beobachten, wie die "Welt" in Form eines hier 1-dimensionalen Grauwertverlaufes an uns vorbeiwandert. Nehmen wir an, die Verschiebung von Frame zu Frame ist so groß, dass eine Diskontinuität im Zeitraum von t-1 bis t am Beobachtungsort x vorbeiwandert. Die Ableitungen I_t und I_x passen dann nicht mehr zusammen: Im Beispiel ist der Quotient $-I_t/I_x$ viel zu groß. Wenn wir aber eine halbwegs brauchbare Schätzung u_c für die Verschiebung haben, dann können wir uns an den kompensierten Ort $x+u_c$ begeben. Dann umfasst die Differenz $I_t^{(c)}=I(x+u_c,t) - I(x,t\text{-}1)$ nicht mehr die Diskontinuität, der Quotient $-I_t^{(c)}/I_x$ liegt viel näher an der Verschiebung u_r (s. Bild), mithin liegt der Term $-I_t^{(c)}/I_x + u_c$ viel näher am wahren Verschiebungsvektor u.

Woher kommt eine "halbwegs brauchbare Schätzung" u_c? – In der Videoanwendigung evtl. von vorherigem Frame-Paar. Wenn es nur das aktuelle Frame-Paar gibt, dann starten wir mit $u_c=0$ und arbeiten iterativ mit einer Methode nach Gl. (6), d.h. wir tasten uns durch bessere Auswertung an anderen Pixeln oder durch Mittelung über die weit streuende Pseudo-Motion aller Pixel schrittweise in die richtige Richtung.

Implementierungsaspekte:

- Auch hier ist die Ableitung $I_t^{(c)}$ in Gl. (7) mit Sub-Pixel-Genauigkeit durchzuführen, also z.B. bilineare Interpolation.
- Man kann einen affinen Parametervektor **a** als Initialschätzung an `pmotion` übergeben (z.B. Ergebnis aus dem vorigen Frame-Paar). Dann wird daraus mit Gl. (6) ein initiales (u_c,v_c) berechnet. Default ist $(u_c,v_c) = 0$.
- Implementierung in <MATLAB>/ima_mos/pmotion.m (Step 1+2) und test_pmotion.m für `itermax>1`:

LS-Schätzung der affinen Parameter

[Die nachfolgende Betrachtung wurde im Grunde schon im Vortrag "Warping" bzw. im Projekt "Image Mosaicing" behandelt, sie ist hier nur der Vollständigkeit halber noch einmal aufgeführt.]

Gegeben ein Pseudo-Motion-Feld (u_p,v_p), welche affinen Motion-Parameter beschreiben es bestmöglich? – Die Antwort zerfällt in zwei entkoppelte Teilprobleme: Suche Parameter (a_1,a_2,a_3) mit

$$a_1 x_i + a_2 y_i + a_3 = u_{pi}$$

bzw. suche Parameter (a_4,a_5,a_6) mit

$$a_4 x_i + a_5 y_i + a_6 = v_{pi},$$

wobei i=1..N über alle Pixel mit `accept==1` läuft. Wir behandeln nachfolgend o.B.d.A. nur das u_p-Problem. Es gibt i.a. viel mehr Gleichungen als Unbekannte, d.h. das Gleichungssystem lässt sich nur im LS-Sinne lösen:

$$(\mathbf{A}\cdot\mathbf{a}-\mathbf{b})^2=\text{Min.}$$

(8) mit $\mathbf{A}=\begin{pmatrix} x_1 & y_1 & 1 \\ \vdots & \vdots & \vdots \\ x_N & y_N & 1 \end{pmatrix}$, $\mathbf{a}=\begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix}$, $\mathbf{b}=\begin{pmatrix} u_{p1} \\ \vdots \\ u_{pN} \end{pmatrix}$

Man beachte, dass **A** und **b** i.d.R. viele Tausend Zeilen lang sein werden! Ein viel einfacheres, nämlich nur 3x3-großes Problem ergibt sich, wenn wir in **Aa=b** beide Seiten von links mit $\mathbf{A}^T$ durchmultiplizieren:

$$(\mathbf{M}\cdot\mathbf{a}-\mathbf{c})^2=\text{Min.}$$

(9) mit $\mathbf{M}=\mathbf{A}^T\mathbf{A}=\begin{pmatrix} \sum x_i x_i & \sum x_i y_i & \sum x_i \\ \sum x_i y_i & \sum y_i y_i & \sum y_i \\ \sum x_i & \sum y_i & \sum 1 \end{pmatrix}$, $\mathbf{c}=\mathbf{A}^T\mathbf{b}=\begin{pmatrix} \sum x_i u_{pi} \\ \sum y_i u_{pi} \\ \sum u_{pi} \end{pmatrix}$

Dabei laufen die Summen über alle i=1..N mit `accept==1`. Deshalb kann sich die Matrix **M** von Iteration zu Iteration ändern, man kann sie (leider!) nicht einmal nur zu Beginn rechnen.

Beide Probleme lassen sich mit der SVD-Inversen zu **A** bzw. **M** (oder in MATLAB mit dem Backslash-Operator) (JAMA: `M.solve(Matrix c)`) lösen.

Implementierungsaspekte:

- Einbau in Step 3 von pmotion.m durch Aufruf von LS_affine.m.

Fragen zu [Kourogi99]

- Wie geht er mit Pixeln um, die I_x=0 oder I_y=0 haben? – Wahrscheinlich schließt er sie aus, genau wie wir.
- Stören Ausreißer in der Pseudo Motion die affine Schätzung? Wie genau geht Kourogi mit Ausreißern um ("M-estimator"), wie definiert er Ausreißer?
- Kourogi schreibt, dass er bei der Implementierung nur N=2000 Pixel verwendet habe. Wie hat er die aus der viel größeren Menge aller Pixel ausgewählt? Random? Oder nur jedes n-te Pixel?
- Was ist "frame-to-mosaic image registration"? – Hier will Kourogi ein (kleines) Videoframe in ein großes Panorama einbauen. Da er beim 1. Frame nicht weiß, wo im Panorama, probiert er zahlreiche Positionen aus und nimmt die mit kleinstem MSE. Ist für uns nicht relevant, wenn wir 'nur' aus konsekutiven Videoframes ein Panorama aufbauen wollen.

Literatur

[Kourogi99] http://citeseer.ist.psu.edu/253440.html: M. Kourogi, T. Kurata, J. Hoshino, and Y. Muraoka. *Real-time image mosaicing from a video sequence*. In Proc. ICIP99,vol. 4, 133--137, 1999.

[HornSchunck81] B. K. P. Horn and B. G. Schunck, *Determining optical flow*, Artificial Intelligence, **17**, 1-3, pp. 185--203, 1981.

[KonBS07] W. Konen, B. Breiderhoff, M. Scholz: Real-time image mosaic for endoscopic video sequences, submitted to BVM (Bildverarbeitung für die Medizin), München, 2007.

Anhang: Die .m-Files in <MATLAB>/real_mos

checkInMask: return true, if floor(rp) accesses a '1' pixel in mask
cmp_amats: skript for generating Fig. 1, 2 in RealTime-ImaMos.doc
cmp_atrue: compare true with estimated affine parameters for each frame
LS_affine: least square estimate of affine parameters
LS_affine2: least square estimate of affine parameters - big matrices
make_figs: skript to make figures for paper2.doc (BVM2007)
mk_masks: create standard masks for rectangular images
mk_masks_endo: create masks for non-rectangular regions
m_xmatymat: make coordinate matrices xmat and ymat + origin [cntx cnty]
panoim_cmp: skript for comparing 'panoim' with original 'imp'
@pano_affine/pano_affine: construct panoramic paint object w
@pano_affine/extend: extend panoramic image by image newim
pmotion: pseudo motion calculation + compensated motion iteration + affine estimate
show_both.m: skript for showing both, endoscope video & panorama video
skript_pmotion: make a simple affine transform frame pair
skript_video: make a simulated endoscope-like video
skript_zoom: make a zoomed endoscope-like frame pair
test_affine: skript for testing LS_affine.m
test_mask: skript for comparing checkInMask & checkInMas2
test_pmotion: skript for testing pmotion.m with rectangular images
test_pmotion1: skript for Table 1 in RealTime-ImaMos.doc
test_pmotion2: skript for Table 2 in RealTime-ImaMos.doc
test_video: skript for testing panorama generation from video
test_zoom: skript for testing panorama generation from zoom pair

Die wichtigsten Files (Endprodukte) aus diesem Verzeichnis sind

- skript_video.m, mit dem sich ein synthetisches Kurz-Video erstellen läßt (einmal aufrufen, es werden Bilddateien auf Platte geschrieben)
- test_video.m: Starter-Skript, lädt Frames eines konsekutiven Videos (z.B. von skript_video.m generiert) von Platte und ruft im wesentlichen in der for-Schleife die Hauptroutinen pmotion.m und @pano_affine\extend.m auf. Ergebnis ist ein Mosaikbild aller Frames aus dem Video.
- pmotion.m berechnet Pseudo Motion und Compensated Motion, benutzt dabei die Hilfsroutinen m_xmatymat.m und LS_affine.m.
- @pano_affine enthält ein Klassenobjekt, mit dem zahlreiche, durch bekannte affine Transformation verbundene Frames zu einem Gesamt-Panorama zusammengesetzt werden können.

Alles andere sind

- Debug-Hilfsroutinen
- oder Visualisierungsroutinen wie panoim_cmp.m und show_both.m[2]
- oder Diagramm- oder Tabellen-Generatoren wie cmp_amats.m (mit Hilfsroutine cmp_atrue.m), make_figs.m, test_pmotion1.m, test_pmotion2.m.

[2] die nach Durchlauf von test_video.m aufgerufen werden können