Implementierung eines Echtzeitverfahrens zur Erstellung von Bildmosaiken aus endoskopischen Videosequenzen

Martin Naderi
mnaderi@gmxpro.de

Fachhochschule Köln
University of Applied Sciences Cologne
Fachbereich Informatik
Studiengang Medieninformatik

Abstract. With image mosaicing[1], it is possible to build a complete Mosaic from many images of a sequence, which shows the content of all images without any redundancy.
The goal of this thesis is to develop an expandable, Object-Oriented software, which processes image data in real-time. This software is based on an existing mosaic algorithm.

1 Einleitung/Zielsetzung

Bei endoskopischen Operationen sind die Einsatzgebiete der digitalen Bildverarbeitung vor allem im Bereich der digitalen Bildoptimierung anzutreffen. So finden sich häufig Arbeiten[2], die durch Bildverbesserungen versuchen die Störeffekte, wie Reflektionen oder Überbelichtungen, von endoskopischem Bildmaterial zu beseitigen. Eine weitere Problemstellung, der bisher aus Sicht der Bildverarbeitung noch keine[3] Aufmerksamkeit geschenkt wurde, ergibt sich durch das beengte Sichtfeld, das dem Chirurg bei der Durchführung einer endoskopischen Operation zur Verfügung steht. Durch diesen Umstand kann es zu Orientierungsproblemen des Chirurgen kommen.
Aus diesem Grund wäre es nützlich und wünschenswert, eine bildverarbeitende Software zur Verfügung zu haben, die automatisch und in Echtzeit (d.h. 5 fps und mehr), aus endoskopischen Bildern ein Panorama bildet. Es existieren viele Algorithmen im Bereich des Image Mosaicing, allerdings arbeiten nur wenige von ihnen vollautomatisch[4] bzw. in Echtzeit[5]. Der in dieser Arbeit verwendete Algorithmus[6] basiert auf der Methode von [Kou99], und erfüllt diese Kriterien.
Die Software soll als ImageJ-Plugin[7] umgesetzt werden. Hierbei besteht die Möglichkeit auf eine bestehende, nicht echtzeitfähige Matlab[8] - Implementierung zugreifen zu können. Für die

[1] [Jac03], [KK04]
[2] Beispielsweise [FVL04]
[3] Mit Ausnahme von [SLH06]
[4] U.a. [Kou99], [Sze94], [SLH06], [Rob03]
[5] [Kou99], [SLH06], [Rob03]
[6] [BKS07]
[7] ImageJ – Java-basiertes Bildverarbeitungsprogramm, das primär in der medizinischen und biologischen Bildverarbeitung eingesetzt wird.
[8] MATLAB – Software die zur Lösung mathematischer Probleme und zur graphischen Darstellung der Ergebnisse eingesetzt wird. Sie ist für Berechnungen mit Matrizen ausgelegt.

Echtzeitfähigkeit des Plugins muss neben der reinen Realisierung natürlich noch die Optimierung des Codes erfolgen, um die Performance möglichst stark zu steigern.
In Kapitel 2 wird kurz auf das grundlegende Verfahren von Kourogi eingegangen, gefolgt von Kapitel 3, das die Umsetzung dieses Verfahrens aufzeigt. Kapitel 4 zeigt schließlich das Ergebnis der Performanceerhöhung .
Der Fokus dieser Zusammenfassung liegt bewusst nur auf dem eigentlichen Verfahren, bzw. auf der Methode *Proc.motion*, da es sich hierbei um den wichtigsten Teil der Arbeit handelt. Die Vorverarbeitung und die Bildung des Panoramas werden hier nicht erwähnt.

2 Grundlagen des Verfahrens

Das Verfahren, das Kourogi[9] zur Berechnung des globalen Bewegungsfeldes zwischen zwei Bildern anwendet, ist in mehrere Schritte unterteilt. Auf diese soll im Folgenden kurz eingegangen werden.
Als Ausgangspunkt für das weitere Vorgehen dient Gleichung (1). Sie bedeutet, dass die Lichtstärke, die zum Zeitpunkt *t-1* in Bildpunkt *(x,y)* war, zum Zeitpunkt *t* nun in Bildpunkt *(x+u, y+v)* ist.

$$I\left(x+u, y+v, t\right)-I\left(x, y, t-1\right)=0 \tag{1}$$

2.1 Pseudo Motion

Da der Verschiebungsvektor *(u,v)* nicht bekannt ist, werden so genannte Pseudo Motion Vektoren errechnet. Sie stellen eine grobe Schätzung des optischen Flusses an jedem Bildpunkt dar. Um die Pseudo Motion Vektoren berechnen zu können, wird bei der Berechnung von *u* der *v*-Term zu Null gesetzt. Nach dem gleichen Schema gestaltet sich die Berechnung von *v*. Somit ergeben sich die folgenden Gleichungen zur Berechnung der Vektoren:

$$\begin{aligned} u_p &= -I_t / I_x \\ &\text{und} \\ v_p &= -I_t / I_y \end{aligned} \tag{2}$$

Bei *Ix, Iy* und *It* handelt es sich hierbei um die partiellen Ableitungen von *I* nach *x, y* und *t*.
Da bei der Errechnung der Motion Vektoren auch unsinnige Werte vorkommen können, wird ein Test (Gleichung (3)) für jeden Pseudo Motion Vektor durchgeführt. Nur die Vektoren die diesen Test passieren werden zur weiteren Berechnung verwendet,

$$\left|I\left(x+u_p, y+v_p, t\right)-I\left(x, y, t-1\right)\right|<T \tag{3}$$

wobei *T* eine geeignet zu wählende Grauwertschwelle ist.

[9] Siehe [Kou99], [Kon06]

2.2 Compensated Motion

Da sich mit der Pseudo Motion nur Verschiebungen von etwa einem Pixel detektieren lassen (bei größeren Verschiebungen wird das Ergebnis sehr ungenau), wird in [Kou99] die Verwendung von der Compensated Motion vorgeschlagen.
Hierfür wird eine grobe Schätzung (u_c,v_c) des Verschiebungsvektorfeldes benötigt, und an allen Stellen eingesetzt. Anschließend wird der verbleibende Rest (u_r,v_r) zum tatsächlichen Vektor (u,v) berechnet. Hierfür muss jetzt natürlich nur noch eine kürzere Strecke geschätzt werden.
Um eine grobe Schätzung für die Compensated Motion zu bekommen, wird nun angenommen, dass das komplette Verschiebungsfeld z.B. aus der Klasse der affinen Transformationen stammt. Somit kann aus der Pseudo Motion dann, unter Verwendung der Least-Square[10] Methode, eine globale Compensated Motion geschätzt werden:

$$\begin{pmatrix} u_c \\ v_c \end{pmatrix} = \begin{pmatrix} a_1 x + a_2 y + a_3 \\ a_4 x + a_5 y + a_6 \end{pmatrix} \tag{4}$$

Die geschätzte Bewegung kann nun zum Zeitpunkt t kompensiert werden. Dies führt zu einer anderen Berechnung von I_t und der Pseudo Motion:

$$I_t^{(c)} = I\left(x+u_c, y+v_c, t\right) - I\left(x, y, t-1\right) \tag{5}$$

und

$$u_p = \left(-I_t^{(c)} / I_x\right) + u_c$$
$$v_p = \left(-I_t^{(c)} / I_y\right) + v_c \tag{6}$$

Die neuen Vektoren müssen wieder nach Gleichung (3) getestet werden. Die abwechselnden Berechnungen von (u_c,v_c) und (u_p,v_p) müssen nun iteriert werden, und führen zu einer besseren Schätzung des Verschiebungsvektorfeldes.

3 Implementierung

Es soll nun auf die Umsetzung des oben beschriebenen Verfahrens, anhand eines Flussdiagramms eingegangen werden. Dieses Diagramm zeigt den Ablauf der Methode *Proc.motion*.
Zu Beginn werden die Gewichtungsfaktoren *wx* und *wy* angelegt, falls es sich um den ersten Aufruf der Methode handelt bzw. falls sich die Bildgröße verändert haben sollte. Die Werte von w*x* und *wy* werden später bei der Berechnung der Abbruchbedingung gebraucht.
Als nächstes führt der Aufruf der Methode *dFilter* dazu, dass die partiellen Ableitungen *Ix* und *Iy* des Referenzbildes errechnet werden.

[10] Die Methode der kleinsten Quadrate ist das mathematische Standardverfahren zur Ausgleichungsrechnung (siehe [Jäh05])

Bevor nun die iterative Berechnung der Pseudo Motion stattfindet, werden *uc* und *vc* mit den affinen Parametern aus dem vorherigen Bildpaar (oder Null wenn es sich um das erste Bildpaar handelt) initialisiert.

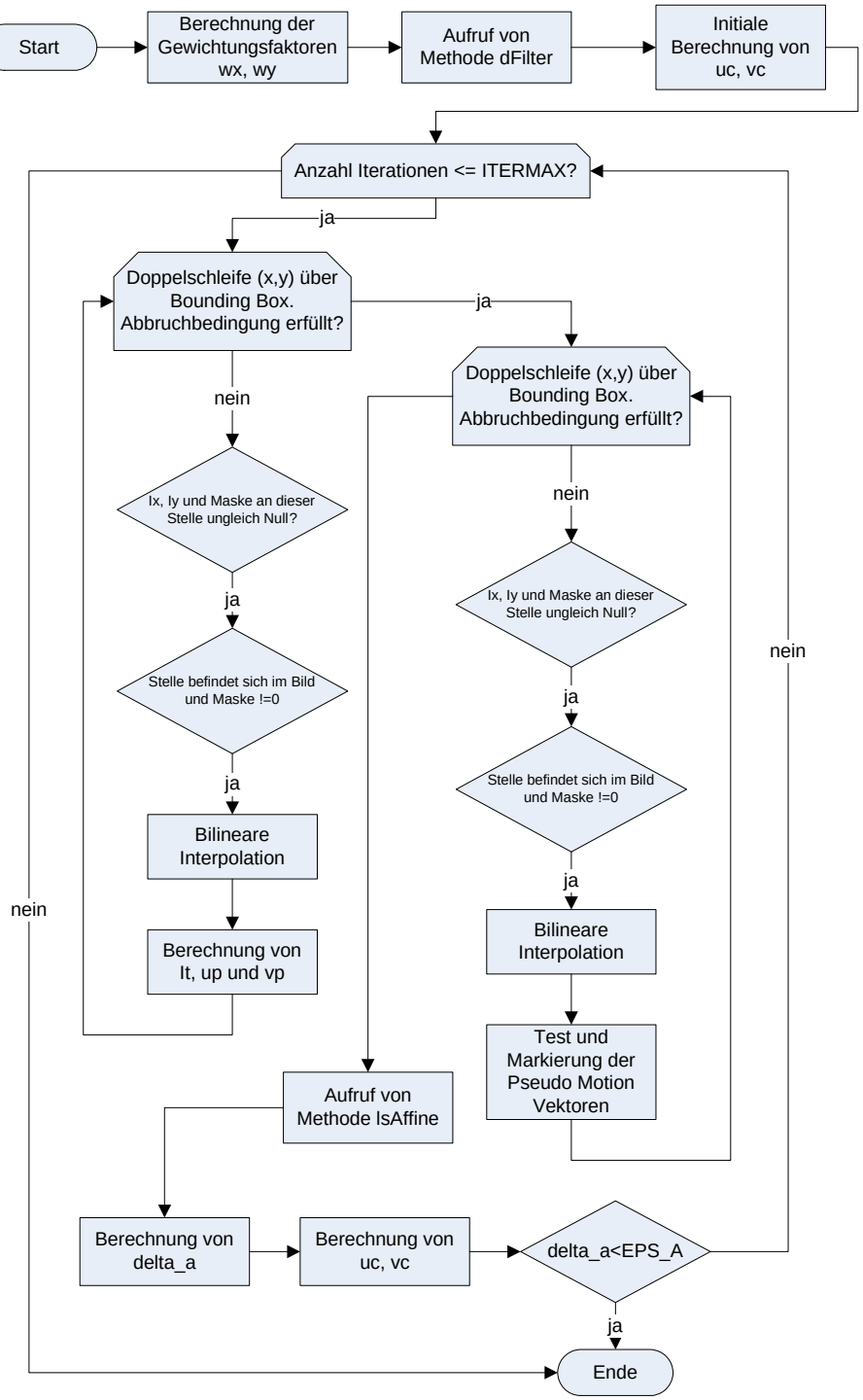

Abbildung 1: Flussdiagramm zur Methode *Proc.motion*

In der ersten Doppelschleife findet nun die Berechnung von *It*, *up* und *vp* statt, wobei die Berechnung von *It* mit Subpixel-Genauigkeit durchgeführt wird (hier: bilineare Interpolation). Durch die zwei Abfragen werden nur die Pixel berechnet, bei denen zum einen *Ix*, *Iy* und die

Bildmaske des Referenzbildes keinen Nullwert aufweisen, und zum anderen die grobe Schätzung (*uc*, *vc*) nicht den Bildbereich bzw. die Bildmaske des neuen Bildes verlässt.
Die gleichen Abfragen finden sich auch in der zweiten Doppelschleife wieder. Hier wird der Test nach Gleichung (3) durchgeführt (ebenfalls mit Subpixel-Genauigkeit), und jene Vektoren die den Test passiert haben, werden in einem Array *accept* markiert.
Der Aufruf der Methode *lsAffine* dient zur Errechnung der 6 affinen Motion Parameter. Danach findet die Berechnung des Wertes von *delta_a* statt. Er gibt an, ob die globale Motion von einer Iteration zur nächsten nur noch gering ist (<*EPS_A*). Ist das der Fall, kann die Berechnung der affinen Parameter für dieses Bildpaar beendet werden. Andernfalls wird solange iteriert, bis die festgelegte Anzahl an Wiederholungen erreicht ist (Flussdiagramm: ITERMAX). Bevor diese Abbruchbedingung allerdings geprüft wird, ist es noch nötig *uc* und *vc* aus den neuen affinen Parametern zu errechnen, um somit für die nächste Iteration wieder eine grobe Schätzung zur Verfügung zu haben.

4 Performance

Neben der Implementierung und Umsetzung des Matlab Codes, ist das Performance-Tuning des Plugins einer der zentralen Punkte dieser Arbeit. Die Optimierung des Codes erfolgte schrittweise während, sowie nach der Implementierung.
Das Ergebnis dieser Optimierungsschritte an der Methode *Proc.motion* ist eine Beschleunigung um den Faktor 22,04, bzw. eine Steigerung der Performance um 2204,32 %.

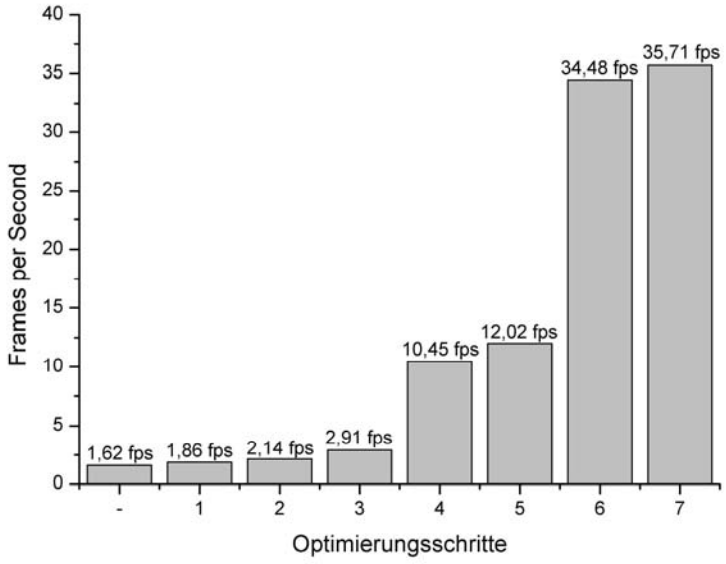

Abbildung 2: Auflistung der Performance nach jedem Optimierungsschritt (*Proc.motion*)

Aus Abbildung 2 ist zu entnehmen, dass vor allem die Schritte 4 und 6 für diese Steigerung verantwortlich sind, wobei in Schritt 4 der Austausch von unperformanten Jama-Methoden zu nennen ist. Die häufigen Exception-Tests und Bounds-Checks bei Zugriffen auf die Matrizenobjekte, machten eine komplette Eigenimplementierung der Methoden erforderlich.
In Schritt 6 wurde der Gewinn an Performance durch Einschränkung der Berechnungen von *uc*, *vc* bzw. *up*, *vp* mittels einer Bounding Box erzielt.

5 Zusammenfassung und Ausblick

Im Zuge dieser Arbeit wurde aus einer bestehenden, nicht echtzeitfähigen Matlab-Implementierung, ein echtzeitfähiges, ausbaubares ImageJ-Plugin realisiert. Im Vordergrund stand, neben der reinen Implementierung in Java, die Optimierung der Performance. Hierdurch wurde, im Vergleich zur ersten, unoptimierten Version, eine Beschleunigung um den Faktor 22 erzielt.
Es wäre für die Zukunft natürlich wünschenswert, den Funktionsumfang der Software um einige sinnvolle Punkte zu erweitern:

- Aufteilung von verschiedenen Threads (Multithreading) auf mehrere Prozessoren. Somit können die Vorteile von heutigen Multicore-Prozessoren genutzt werden.
- Die Erweiterung, von der hier verwendeten affinen, auf die projektive Transformation ermöglicht es, mehrere Kamerabewegungen abzubilden.
- Der Einsatz von Algorithmen, die robust auf Beleuchtungsänderungen reagieren. So können beispielsweise sogenannte Glanzlichter, durch Reflexionen des Endoskoplichts auf feuchten Organoberflächen, entstehen, und die Qualität des Bildes verschlechtern.
- Die Vermeidung von Bildverzeichnungen im Panoramabild, durch Verzeichnungskorrektur, wobei die Verzeichnungen durch die eingesetzte Kameraoptik entstehen.
- Die Software sollte automatisch Schnitte im Videostream erkennen können.

6 Literatur (Auswahl)

[BKS07] *Breiderhoff B., Konen W., Scholz M., Ein automatisiertes Verfahren zum Image-Mosaicing bei endoskopischen Videoaufnahmen*, Technical Report, Inst. for Informatics, FH Cologne, 2007

[FVL04] *Fischer B, Vaessen B, Lehmann TM, Spitzer K*, Bildverbesserung endoskopischer Videosequenzen in Echtzeit, 2004

[Jac03] *Jacobs A.*, Mosaicing auf Szenen mit bewegten Objekten, Diplomarbeit, 2003

[Jäh05] Jähne, B.: Digitale Bildverarbeitung, Springer Verlag, Berlin, 2005.

[KK04] *Klein, Hans-Ulrich / Koop, Michael:* Image Mosaicing, Seminararbeit,Westfälische Wilhelms-Universität Münster, Institut für Informatik, Lehrstuhl Prof. Dr. Xiaoyi Jiang, 2004

[Kon06] *Konen W*, Optischer Fluss und Echtzeitvideobearbeitung, Technical Report, Inst. For Informatics, FH Cologne, 2006

[Kou99] *Kourogi M, Kurata T, Hoshino J, et al.*: Real-time image mosaicing from a video sequence. Procs ICIP99, vol. 4, 133-137, 1999.

[Rob03] *Robinson JA*: A Simplex-Based Projective Transform Estimator. Procs Visual Information Engineering (VIE), Guildford, 290-293, July, 2003.

[SLH06] *Seshamani S, Lau WW, Hager GD*: Real-Time Endoscopic Mosaicking. MICCAI'2006, Lecture Notes in Computer Science, vol. 4190, Springer, 355-363 2006.

[Sze94] *Szeliski R*: Image Mosaicing for Tele-Reality Applications. TR 94/2, Digital Equipment Corporation, Cambridge Research Lab, June 1994.