
Technische Universität Wien

Vienna University of Technology

universität wien

# Masterarbeit

Titel der Masterarbeit

## Bewegungsanalyse von Videos unter Berücksichtigung von Verdeckungen

Verfasser

Bakk. Christian Ammer

angestrebter akademischer Grad
Magister der Sozial- und Wirtschaftswissenschaften (Mag.rer.soc.oec)

Wien, 2009

Studienkennzahl lt. Studienblatt: A 066 926
Studienrichtung lt. Studienblatt: Wirtschaftsinformatik
Betreuerin: Mag. Dipl.Ing. Dr. Margrit Gelautz, ao. Univ. Prof.

**Abstract**

Motion estimation is used to detect motion model parameters and to assign each pixel in a video frame to one of the identified motion models. Motion estimation is a challenging task in computer vision because many applications require high qualitative motion description of the video. For example, the high compression rates of videos are largely depending on the motion compensation of video streams. The automatic identification of objects and the three dimensional scene reconstruction are also depending on motion estimation.

In this work, we present an algorithm which is based on the work of Jiangjan Xiao and Mubarak Shah. The algorithm analyses a short video sequence. In contrast to previous work, it takes occlusions into account. The result of the computation is a layered representation of the video, whereby each layer represents one affine motion model.

In the first step the algorithm identifies motion parameters. The algorithm selects feature points and uses a region growing method for each feature point. In every growing stage the motion model parameters of the region are re-estimated via the Newton procedure. After this step, similar motions are combined. In the layer assignment step, an energy minimization method is used to minimize a cost function and to assign each pixel to one of the previously identified motion parameters. In addition to common data and smoothness terms, the cost function includes an occlusion term. The energy minimization method is based on graph-cuts.

## Zusammenfassung

Die Aufgabe von Motion Estimation ist, Bewegungsbereiche in einem Video zu erkennen und jedem Bildpunkt eine Bewegung zuzuordnen. Es ist eine herausfordernde Aufgabe von Computer Vision, denn viele Anwendungen des Maschinellen Sehens lassen sich erst durch eine qualitativ hochwertige Bewegungsanalyse des Videos lösen. So sind die hohen Kompressionsraten in Videos mitunter auf die Bewegungskompensation zurückzuführen, welche durch Motion Estimation möglich wird. Eine automatische Erfassung und Identifizierung von Gegenständen oder eine dreidimensionale Szenenrekonstruktion sind Anwendungen, welche eine Bewegungsanalyse voraussetzen.

In dieser Arbeit wird ein Algorithmus auf Grundlage der Arbeiten von Jiangjan Xiao und Mubarak Shah entworfen, der eine kurze Videosequenz analysiert. Im Gegensatz zu anderen Arbeiten in diesem Bereich berücksichtigt der Algorithmus Verdeckungen. Das Ergebnis einer Analyse ist eine Ebenenrepräsentation des Videos, bei der jeder Ebene eine affine Bewegung zugrunde liegt.

Um die affinen Bewegungen zu finden, sucht der Algorithmus im ersten Schritt Featurepoints. Jeder Featurepoint wird als Saatpunkt im folgenden Region-Growing-Schritt verwendet. In jeder Region-Growing-Wachstumsphase werden die affinen Bewegungen mit dem Newton Näherungsverfahren bis zu einem bestimmten Punkt verfeinert. Ähnliche affine Bewegungen werden zu einer zusammengefasst. Im nächsten Schritt, dem Layer Assignment Schritt, wird eine Energiefunktion entworfen und durch das Minimale-Schnitt-Verfahren minimiert. Durch die Minimierung wird jedem Bildpunkt eine affine Bewegung zugeordnet, was die Lösung des Zuordnungsproblems darstellt. Die Energiefunktion enthält neben Daten- und Smoothnesskosten auch einen Term für Verdeckungen zwischen Bildern und einen Term für Verdeckungen zwischen Bildpaaren. Das Minimale-Schnitt-Verfahren wird seit einigen Jahren häufig für Computer Vision Aufgaben eingesetzt, da es sich in der Praxis gut bewährt.

# Inhaltsverzeichnis

# 1 Einleitung

## 1.1 Aufgabenstellung

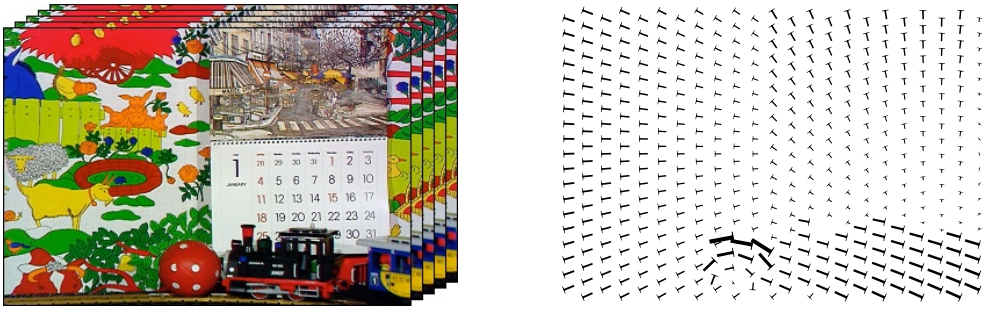
Ein *Bildpunkt* in einem Video wird durch drei Dimensionen identifiziert. Die $x$ Koordinate, die $y$ Koordinate und die Zeitdimension, die sich durch die Nummer des Bildes ausdrückt. Die Idee, jedem Bildpunkt eine Bewegung zuzuordnen, bietet für viele Anwendungen Vorteile. Die Umsetzung dieser Idee wird auch als das Bestimmen des *Optical Flows* bezeichnet [13, 23]. Ein anderer Begriff dafür ist *Motion Estimation*. Arbeiten, die sich mit diesem Thema beschäftigen, sind in das Wissenschaftsgebiet *Computer Vision* einzuordnen. Abbildung 1.1 zeigt das Ergebnis einer Optical Flow Bestimmung einer kurzen Videosequenz.

Die Aufgabe dieser Arbeit war es, einen Algorithmus zu entwerfen, mit dem eine kurze Videosequenz analysiert werden kann und dessen Ausgabe eine Ebenenrepräsentation, einschließlich Bewegungstransformation, der Videosequenz darstellt. Die Ebenenrepräsentation der Bewegungen bietet mehr Information als der Optical Flow, da diese Repräsentation zusätzlich die Objektgrenzen enthält. Die Ebenenbewegungen werden im Folgenden als *Motion Layer*, oder nur *Layer*, bezeichnet. Nach einer Literaturrecherche haben sich die Arbeiten *Motion Layer Extraction in the Presence of Occlusion using Graph Cuts* [35] und *Accurate Motion Layer Segmentation and Matting* [36] von Jiangjan Xiao und Mubarak Shah für die Basis dieser Arbeit als sehr viel versprechend herausgestellt. Ein Merkmal, das ihre Arbeiten gegenüber anderen Arbeiten hervorhebt, ist die ausdrückliche Berücksichtigung von Verdeckungen im Video. Der Algorithmus in dieser Arbeit lehnt sich stark an den Algorithmus an, den J. Xiao und M. Shah vorgeschlagen haben.

## 1.2 Probleme

Einige Phänomene erschweren die Optical Flow Berechnung. So ist es nicht möglich, einen Bildpunkt in einem Bild exakt im nächsten Bild zu erkennen. Beleuchtungsänderungen und Bildrauschen verändern seine Farbe und Intensität. Das macht eine eindeutige Zuordnung schwer möglich. Darum werden die Nachbarbildpunkte immer miteinbezogen. Doch selbst dann ist es schwer, in Bereichen ohne Textur eine Bewegung zu erkennen. Im Gegensatz dazu erzeugen Reflexionen eine Textur, die in Wirklichkeit keine ist.

Ein Gegenstand in einem Video verdeckt Bereiche im Hintergrund. Wenn sich der Gegenstand bewegt, dann werden diese Bereiche sichtbar, dafür verschwinden andere Bereiche (siehe Abbildung 1.2). Zwischen aufeinanderfolgenden Bildern in einem Video



(a) *Mobile & Calendar* Video (b) Quiver Plot

Abbildung 1.1: Aus einer kurzen Sequenz des *Mobile & Calendar* Videos werden die Bewegungsvektoren jedes Bildpunkts berechnet. Die Bewegungsvektoren für das erste Bild sind in (b) in einem Quiver Plot dargestellt.

gibt es also immer Bereiche, die in einem der Bilder nicht sichtbar sind. Diese Bereiche unterliegen zwar einer Bewegung, ein Algorithmus hat aber Schwierigkeiten diese korrekt zuzuordnen.

Ein weiteres Problem ist das *Aperture Problem*. Bei einer Linie kann nur die orthogonale Komponente der Bewegung festgestellt werden. Eine Bewegung entlang der Linie kann nicht ohne weiteres festgestellt werden.

Schatten stellen auch eine Schwierigkeit dar. Sie treten in fast jeder Szene, abhängig von den Beleuchtungsverhältnissen, mehr oder weniger intensiv auf. Wenn ein Schatten eines bewegten Gegenstands auf eine Fläche mit wenig Textur fällt und scharf abgegrenzt ist, ist eine korrekte Zuordnung sehr schwierig. So wird der Schatten eines Gegenstands oft dem Gegenstand selbst zugeordnet, statt dem Gegenstand, auf dem er fällt.

## 1.3 Anwendungen

Die hohen Kompressionsraten bei Videos sind unter anderem auf Bewegungskompensationen zurückzuführen. Ein Differenzbild mit Bewegungskompensation hat gegenüber einem Differenzbild ohne Bewegungskompensation wesentlich weniger Informationen. Das reduziert die Datenmenge erheblich, Abbildung 1.3 veranschaulicht diesen Gewinn durch die Bewegungskompensation.

Die Qualität digitaler Videos hat mit der Kapazität der Datenträger als auch mit den verbesserten Kompressionsverfahren ständig zugenommen. In letzter Zeit wird sogar Fernsehen digital mit dem DVB Standard übertragen. Die *Moving Picture Experts Group* – MPEG – hat mehrere Standards zur Videokompression verabschiedet. Während eine Video CD im Format MPEG-1, eine DVD im Format MPEG-2 aufgenommen wird, kommt bei der Blu-ray Disc entweder MPEG-2, H.264 oder VC-1 zum Einsatz. Das H.264 Format – MPEG-4/AVC – ist das effizienteste Videokompressionsverfahren zurzeit. Gegenüber seinen Vorgängern verwendet H.264 Bewegungsvektoren mit einem viertel, statt einem ganzen oder halben Bildpunkt Genauigkeit [33]. Das H.264 Video-

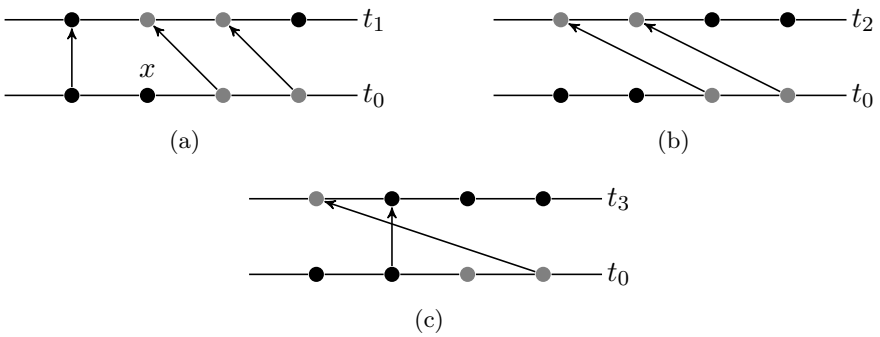



Abbildung 1.2: Ein Objekt – durch die grauen Punkte symbolisiert – bewegt sich vor einem unbewegten Hintergrund – durch die schwarzen Bildpunkte symbolisiert – von rechts nach links. Die Zuordnungen der Punkte zwischen zwei Bildern werden durch Pfeile dargestellt. Punkte, die mit keinem Pfeil verbunden sind, werden durch das Objekt verdeckt. Der Punkt $x$ in Darstellung (a) ist zum Zeitpunkt $t_0$ verdeckt, da sich zum Zeitpunkt $t_1$ das Objekt vor diesen Punkt geschoben hat. Dieser Punkt wird erst in Darstellung (c) wieder sichtbar.

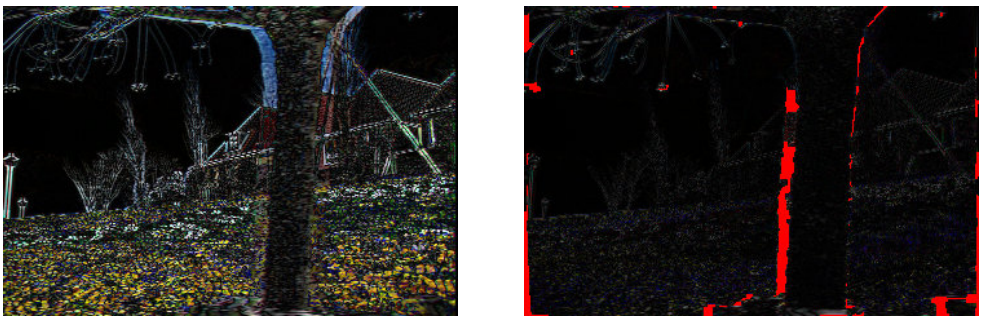



Abbildung 1.3: Vergleich von Differenzbildern der Sequenz *Flower Garden*. Bild (a) zeigt die Differenz von Bild 1 und 3 ohne Bewegungskompensation, Bild (b) mit Bewegungskompensation. Die roten Bereiche in (b) zeigen verdeckte Bereiche.

kompressionsverfahren wurde auch als Teil in den MPEG-4 Standard aufgenommen.

Mit dem MPEG-4 Standard wurde auch die Möglichkeit geschaffen, eine Szene aus Objekten – so genannte *media objects* – zusammenzustellen. Diese Objekte können Bilder, Videoobjekte und Audioobjekte sein. Dadurch lassen sich die Objekte einfacher manipulieren. Ein weiterer Vorteil ist, dass die einzelnen Objekte entsprechend ihrem Format effizienter gespeichert werden können. Um ein Video im MPEG-4 Format speichern und die Vorteile der *media objects* nutzen zu können, bedarf es Algorithmen, welche die einzelnen Motion Layer aus der Orginalaufnahme extrahieren.

Ist die Szene im Video statisch und die Kamerabewegung bekannt, lässt sich die Szene dreidimensional rekonstruieren. Diese Rekonstruktion birgt Potenzial für weitere Anwendungen, wie das *Image Based Rendering*.

Roboternavigation, Objekterkennung oder die automatische Beschreibung von Videosequenzen sind weitere Anwendungsmöglichkeiten. Motion Estimation ist also eine Basistechnik, auf welcher die genannten höheren Anwendungen aufbauen.

## 1.4 Literaturvergleich

Einen guten Einstieg in die Thematik bietet die Arbeit *Representing Moving Images with Layers* [32] von John Y. A. Wang und Edward H. Adelson. Die Arbeit ist bereits 15 Jahre alt, sie gibt aber einen guten Einblick in die Idee, ein Video zu analysieren und die einzelnen Bewegungsbereiche als unabhängige Layer zu betrachten. Jeder Layer besteht aus einem statischen Bild, inklusive einem Kanal für die Transparenz. Jeder Layer hat eine bestimmte Z-Ordnung, die angibt, wie die Layer übereinandergelegt sind. Jeder Layer hat eine zugrunde liegende affine Bewegung, die sich von Bild zu Bild ändert. In ihrer Arbeit analysieren sie 30 Bilder der Testsequenz *Flower Garden*, aus denen sie 3 Layer bilden – die Häuser im Hintergrund, den Baum im Vordergrund und das Blumenbeet dazwischen. Die Layer können danach wieder kombiniert abgespielt werden. Es kann aber auch ein Layer, wie zum Beispiel der Baum im Vordergrund, weggelassen werden.

Die Aufgabe von *Motion Estimation* kann in zwei Hauptaufgaben getrennt werden. Das Erkennen der Bewegungen und das Zuordnen dieser Bewegungen zu den einzelnen Bildpunkten. Für das Erkennen der Bewegungen verwenden John Wills, Sameer Agarwal und Serge Belongie den RANSAC Algorithmus in einem zweistufigen Verfahren. Ihre Methode, die in [34] genauer beschrieben wird, läuft folgendermaßen ab: Es werden Featurepoints mit dem Förstner Operator identifiziert. Jeder Featurepoint wird dann mit Hilfe von 76 Filtern beschrieben. Die Filter bestehen unter anderen aus Filtern, die ein Fenster zu 3 Stufen skaliert und zu 12 verschiedenen Winkeln rotiert. Die Filter werden auf ein bis maximal $31 \cdot 31$ Pixel großes Fenster angewandt. Die Übereinstimmung der Featurepoints zwischen zwei Bildern wird dadurch gefunden, dass die Filtervektoren mit der $L_1$ Distanz verglichen werden. Dabei wird jeder Featurepoint aus dem einen Bild mit jedem Featurepoint aus dem anderen Bild verglichen. Es korrespondieren immer jene Paare mit der kleinsten $L_1$ Distanz zueinander. In einem ersten RANSAC Schritt werden die Featurepoints grob gruppiert und eine Ebenenbewegung zu jeder Gruppe ge-

schätzt. Die Gruppen werden nach der Anzahl der beinhaltenden Featurepoints sortiert. Von Gruppen, die sich mehr als 75% überdecken, wird nur die Gruppe mit der größeren Anzahl an Featurepoints behalten. In einem zweiten RANSAC Schritt, der diesmal innerhalb jeder Gruppe durchgeführt wird, wird die Ebenenbewegung jeder Gruppe genauer geschätzt. Im Zuordnungsschritt verwenden sie eine Energiefunktion, die einen Datenterm – mit quadratischem Intensitätsabstand – und einen Smoothnessterm, allerdings keine Behandlung von Verdeckungen, beinhaltet.

Ein anderer Weg wird von Qifa Ke und Takeo Kanade gewählt [18, 17]. Sie extrahieren aus den Eingabebildern affine Bewegungen – ein Vektor, der eine affine Bewegung ausdrückt, besitzt 6 Dimensionen –, welche sie in einem dreidimensionalen Unterraum abbilden. In diesem Unterraum rücken ähnliche affine Bewegungen näher zusammen und lassen sich dadurch gruppieren. Der Ablauf ist folgender: In den Eingabebildern werden Segmente gebildet. Jedes Segment im Referenzbild wird in den anderen Eingabebildern gesucht und die dabei stattgefundene affine Bewegung geschätzt. Eine affine Bewegung dient als Referenz, zu dieser werden alle anderen als relative affine Bewegungen berechnet. Danach erfolgt die Abbildung in den 3 dimensionalen Unterraum. Diese ist nur deshalb möglich, weil alle Ebenenbewegungen in einem Bild sich dieselben Kameraeigenschaften teilen. Es bleiben alle Informationen der relativen affinen Bewegungen erhalten, wie in [18, 17] gezeigt wird. In diesem Unterraum werden die Ebenenbewegungen durch einen *Mean Shift* Algorithmus gruppiert. Jede Gruppe bildet einen Layer, zu dem die affinen Parameter neu berechnet werden. Die Eingabebilder werden farbsegmentiert und jedes Farbsegment dem Layer zugeordnet, der am besten seine Bewegung schätzt.
Die Arbeiten [18, 17] unterscheiden sich bei der Segmentbildung: In [18] wird Farbe als Segmentierungskriterium verwendet. In [17] werden *k-connected component* Segmente gebildet. Der Vorteil der *k-connected components* ist, dass Ausreißer relativ früh erkannt und ausgeschieden werden. Außerdem werden in [17] Ausreißer zusätzlich im Unterraum durch die Mahalanobis Distanz identifiziert. Der Algorithmus aus [17] ist durch diese Behandlung der Ausreißer robuster als der Algorithmus aus [18].

Serge Ayer und Harpreet S. Sawhney [1] verwenden die *Maximum-Likelihood-Methode* (MLE) und die *Minimum-Description-Length-Methode* (MDL) um jene Layer zu finden, welche das Video angemessen beschreiben. Für ihre Methode benötigen sie keine videoabhängigen Parameter. Ihr Hauptaugenmerk lag in der präzisen mathematischen Formulierung des Motion Problems. Andere Arbeiten, welche statistische Schätzverfahren wie MLE oder die *Maximum-A-Posteriori-Methode* (MAP) verwenden, sind in [37, 24, 30] dargestellt.

Stereo Vision, ein Computer Vision Teilgebiet, hat eine ähnliche Aufgabe wie Motion. Der Unterschied ist, dass bei Stereo Vision die Kamerapositionen bekannt sind, und die Eingabe nur aus zwei Bildern besteht. Es gilt die Epipolargeometrie, das heißt, ein Bildpunkt wird im zweiten Bild auf der gleichen horizontalen Linie gefunden wie im ersten Bild. Der Bildpunkt ist um eine Disparität verschoben. Wie bei vielen Computer Vision Problemen besteht die Herausforderung in Stereo Vision im Minimieren von Energiefunktionen. Rick Szeliski, Ramin Zabih und Daniel Scharstein haben viele Stereo Algorithmen verglichen [25, 29]. Bei diesen Vergleichen haben sich jene besonders hervorgetan, welche das Verfahren des *Minimalen Schnitts* zum Minimieren angewandt

hatten.

In dieser Arbeit wird versucht, die guten Ergebnisse der Arbeiten von Jiangjan Xiao und Mubarak Shah zu erreichen. Ihre Arbeit haben sie in den Dokumenten *Motion Layer Extraction in the Presence of Occlusion using Graph Cuts* [35] und *Accurate Motion Layer Segmentation and Matting* [36] veröffentlicht. Für das Erkennen der Bewegungen verwenden sie einen Region Growing Algorithmus. Im Zuordnungsschritt definieren sie eine Energiefunktion, welche eine Berücksichtigung von Verdeckungen einschließt. Das ist ein neuartiger Ansatz, da Verdeckungen in Motion Estimation noch kaum untersucht wurden. In Stereo Vision gibt es dagegen einige Arbeiten [3, 4, 20, 21, 16], welche den Verdeckungen in einem Stereo-Bildpaar besondere Aufmerksamkeit schenken.

Ein Stereoalgorithmus, der Verdeckungen berücksichtigt, wurde von Michael Bleyer im Rahmen seiner Dissertation [3] entwickelt. Durch den Verzicht der Epipolargeometrie hat er ihn auch für das Motion Problem erweitern können. Der Stereoalgorithmus hat folgenden Ablauf: Die Eingabebilder werden farbsegmentiert. Mit einer fensterbasierten Suchmethode wird eine erste Disparitätskarte erstellt. Mit Hilfe der Disparitätskarte wird für jedes Segment ein Ebenenmodell berechnet. Danach werden die Ebenenmodelle mit einer *Mean-Shift-Methode* zu Layer zusammengefasst. Im letzten Schritt, dem Layer Assignment Schritt, wird versucht das Ergebnis zu verbessern. Verschieben von Layer-Randsegmenten zu benachbarten Layern kann eine Qualitätszunahme, welche mit einer Kostenfunktion gemessen wird, bewirken. Im Layer Assignment Schritt werden auch die verdeckten Bildpunkte bestimmt.

Dass der Algorithmus die Eingabebilder farbsegmentiert, bietet einige Vorteile. Unter der Annahme, dass ein Farbsegment einem Disparitätsmodell zugrunde liegt, kann das Problem der untexturierten Bereiche – innerhalb welcher es schwer ist Disparitäten zuzuordnen – umgangen werden. Außerdem werden Tiefenunstetigkeiten an Objektkanten durch Farbsegmentierung oft besser erkannt als dies durch ausschließliche Berücksichtigung von Disparität möglich wäre.

## 1.5 Aufbau der Arbeit

Diese Arbeit ist folgendermaßen aufgebaut: Das Kapitel 2 erläutert die Begriffe und Techniken, die eingesetzt wurden. Es wird der mathematische Begriff einer affinen Abbildung erklärt. Es wird darauf eingegangen, wie Featurepoints identifiziert und über mehrere Bilder hinweg verfolgt werden können. Der *Minimale Schnitt* und das *Expansion Move* Verfahren sind weitere Techniken, die in diesem Kapitel ausführlich erklärt werden.

Das Kapitel 3 erläutert den Algorithmus. Es beginnt mit einem einführenden Beispiel, das einen Überblick über den Algorithmus geben soll. Danach wird in den zwei Unterkapiteln 3.2 und 3.3 auf die einzelnen Schritte eingegangen.

Die Implementierung des Algorithmus ist im Kapitel 4 dargelegt. Der Aufbau vom Programm, die benutzten Klassen und Funktionen und die Programm Ein- und Ausgaben sind der Inhalt dieses Kapitels. Das Programm wird mit verschiedenen Testsequenzen getestet und die Ergebnisse werden im Kapitel 5 dargestellt und erläutert.

In der Zusammenfassung im Kapitel 6 wird auf die Vorzüge und Nachteile des Programms und des Algorithmus hingewiesen, die sich bei der Beurteilung der Ergebnisse herausgestellt haben. Weiters wird auf Ideen und Möglichkeiten eingegangen, wie diese Arbeit fortgesetzt werden könnte.

# 2 Grundlagen

In diesem Kapitel wird auf die Grundlagen eingegangen. Im ersten Unterkapitel 2.1 wird der Begriff *affine Abbildung* erklärt. Dieser Begriff ist zentral und wird in dieser Arbeit häufig, auch unter dem Synonym *affine Bewegung*, verwendet. Ein Bildpunkt wird nicht für sich allein, sondern in Abhängigkeit von seinen Nachbarn betrachtet. Nachbarbildpunkte sind, wenn sie zum selben Objekt in der Wirklichkeit gehören, aneinander gekoppelt. Denn es sind immer ganze Bereiche in einem Bild, die einer Bewegung – der des Objekts, bzw. der Kamera – zugrunde liegen. Bildpunkte, die derselben Bewegung unterliegen, können zu Ebenen zusammengefasst werden. Diese Ebenen gehorchen dann einer Ebenenbewegung. Die Ebenenbewegung wird durch eine *affine Bewegung* beschrieben.

Im Unterkapitel 2.2 wird auf die Schwierigkeit eingegangen, einen Bildpunkt in einem Video zu verfolgen. Weiters wird beschrieben, wie die affinen Parameter berechnet werden. Das Unterkapitel 2.3 erklärt, was eine Energiefunktion ist, wie eine solche Funktion aufgebaut ist und wie sie helfen kann, eine Aufgabe aus Computer Vision zu lösen. In den beiden letzten Unterkapiteln 2.4 und 2.5 wird auf das Minimieren von Energiefunktionen eingegangen. Dieser Schritt wählt aus den vielen Lösungen der Energiefunktion eine Lösung aus, die ein lokales Minimum darstellt – da es sich um ein Näherungsverfahren handelt, wird das globale Minimum wahrscheinlich nicht erreicht.

## 2.1 Affine Abbildung

Ein simples mathematisches Modell, um die Bewegung einer Region im Bild zu beschreiben, ist das Translationsmodell. Bei diesem Modell bewegen sich alle Bildpunkte einer Region in die gleiche Richtung mit immer dem gleichen Abstand. Wenn sich ein Objekt von der Kamera fortbewegt, kann das Translationsmodell nicht mehr für alle Bildpunkte dieser Region gelten, da sich eine derartige Bewegung als Skalierung ausdrückt. Bei einer Skalierung ist die Richtung der Bewegungsvektoren zum Zentrum hingerichtet oder aus dem Zentrum kommend. Somit haben die Bewegungsvektoren unterschiedliche Richtungen und auch unterschiedliche Längen. Ein besseres Modell zur Beschreibung von Bewegung ist die *affine Abbildung*. Eine affine Abbildung kann eine Kombination aus Skalierung, Rotation, Scherung, Reflexion und Translation modellieren.

Lineare Abbildung und affine Abbildung sind Begriffe aus der linearen Algebra. Eine lineare oder affine Abbildung ist eine Funktion, die einen Vektor aus einem Vektorraum in einen anderen Vektorraum abbildet. Eine lineare Abbildung kann Rotation, Scherung, Skalierung und Reflexion abbilden. Eine affine Abbildung ist eine lineare Abbildung mit Translation. Bei der affinen Abbildung bleiben Abstandsverhältnisse erhalten und Punkte auf einer Geraden liegen nach der Abbildung weiterhin auf einer Geraden.

## 2.2 Features Finden und Tracken

Weil nicht jeder Bildpunkt zweifelsfrei verfolgt werden kann, werden nur ausgewählte Bildpunkte, so genannte *Featurepoints*, verfolgt. Diese haben die Eigenschaft, dass sie in einem Bild leicht identifiziert und über die nächsten Bilder verfolgt werden können. Aus diesem Grund sind Arbeiten, die vom *Tracken* von Features handeln, für Motion Estimation von Interesse. Takeo Kanade, Carlo Tomasi und Jianbo Shi haben einen Feature Tracker entwickelt, der in Computer Vision oft eingesetzt wird [27, 15]. Wahrscheinlich aus diesem Grund wurde er auch in die OpenCV Bibliothek [14] von Intel integriert.

Der *KLT Tracker* wird auch in dieser Arbeit dazu verwendet, Features zu finden und diese über die gesamte Sequenz zu tracken. KLT steht für die Namen seiner Entwickler Kanade, Lucas und Tomasi. Beschrieben wird der KLT Tracker in den Arbeiten von Shi und Tomasi [27] und Kanade und Tomasi [15].

Zusammengefasst kann man sagen, dass der KLT Tracker im ersten Bild einer Sequenz Features sucht, die relativ eindeutig sind und auch echten Punkten an einem Gegenstand in der Wirklichkeit entsprechen. Ein Feature, das keinem Punkt in der Wirklichkeit entspräche, wäre zum Beispiel eine Reflexion an einem spiegelnden Gegenstand. Der Tracker versucht die ermittelten Features über die ganze Sequenz zu verfolgen. Dabei können die Features verschwinden, wenn das Objekt verdeckt wird, bzw. auch wieder auftauchen.

Am einfachsten kann man sich einen Feature Tracker vorstellen, indem ein Feature nicht als Bildpunkt getrackt wird, sondern als Fenster. Man bewegt das Featurefenster über das zweite Bild und berechnet an jeder Position einen Übereinstimmungswert – zum Beispiel durch die Summe der quadratischen Helligkeitsunterschiede. An der Position, an der der Übereinstimmungswert am größten ist, hat man das Feature im zweiten Bild gefunden.

Der KLT Tracker verwendet eine ausgefeiltere Technik und liefert dadurch bessere Ergebnisse. Er benutzt zwei Modelle zum Tracken der Features, zum einen ein reines Translationsmodell und zum anderen ein affines Bewegungsmodell (Abbildung 2.1). Experimente [27] haben gezeigt, dass bei einer Kombination dieser Modelle die besten Ergebnisse dann erzielt werden, wenn die Features mit dem Translationsmodell getrackt werden und mit dem affinen Bewegungsmodell die Qualität festgestellt wird. Wenn die Qualität nicht mehr ausreicht, wird das Feature ausgeschieden und gegebenenfalls durch ein neues ersetzt.

Im Detail funktioniert der Algorithmus so, dass mit dem Translationsmodell ein Feature von Bild zu Bild getrackt wird. Die Annahme dabei ist, dass die translative Bewegung eines Features von unmittelbar aufeinanderfolgenden Bildern gering, aber messbar ist, eine affine Bewegung jedoch kaum stattfindet. Würde man auch eine affine Bewegung messen wollen, wäre das eher eine zusätzliche Fehlerquelle, als dass es das Ergebnis positiv beeinflussen könnte. Bei weiter auseinander liegenden Bildern aber ist auch eine affine Bewegung messbar. Der Tracker macht sich das zu Nutze und stellt auf diese Weise die Qualität jedes Features fest [27].

Im Folgenden wird auf den Algorithmus detaillierter eingegangen, es gibt dabei zwei Fragen zu klären. Was ein gutes Feature ausmacht, wird im Kapitel 2.2.1 beschrieben.

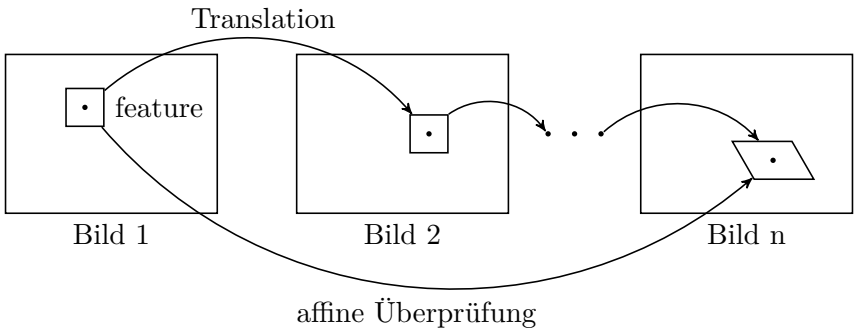



Abbildung 2.1: Ablauf des Trackens eines Featurepoint. Zwischen aufeinanderfolgenden Bildern wird nur eine Verschiebung des Featurefensters berücksichtigt. Zwischen weiter auseinander liegenden Bildern wird die Qualität des Trackingerfolgs mit dem affinen Bewegungsmodell überprüft.

Wie dieses Feature getrackt wird, wird im Kapitel 2.2.2 beschrieben.

### 2.2.1 Berechnung der affinen Parameter

In dieser Arbeit kommt die Berechnung der affinen Parameter an zwei Stellen vor. Zum einen wird die Berechnung im Region Growing Algorithmus (siehe Kapitel 3.2.1) benötigt, zum anderen verwendet sie der KLT Tracker, um die Qualität der getrackten Features zu prüfen. Der Unterschied zwischen der Berechnung, wie sie der KLT Tracker verwendet und wie sie im Region Growing verwendet wird, liegt darin, dass der KLT Tracker ein Fenster für die Berechnung verwendet und im Region Growing aber beliebige Regionen als Basis genommen werden.

In einem Bild findet man ein Feature, wenn es nicht verdeckt wurde, auch im nächsten Bild. Man kann dem Feature eine affine Bewegung zu Grunde legen. Eine affine Bewegung wird mit einer Transformationsmatrix $A$ und einem Translationsvektor $d$ beschrieben (Gleichung (2.1)).

Eine affine Bewegung besteht aus einer Transformation und aus einer Translation. Die Transformationsmatrix hat die Gestalt aus Gleichung (2.2), der Translationsvektor die Gestalt aus Gleichung (2.3).

$$x \mapsto Ax + d \tag{2.1}$$

$$A = \begin{bmatrix} a_{xx} & a_{xy} \\ a_{yx} & a_{yy} \end{bmatrix} \tag{2.2}$$

$$d = \begin{bmatrix} d_x \\ d_y \end{bmatrix} \tag{2.3}$$

Ein Bildpunkt $x$ im Bild $I$ wandert im Bild $J$ an die Position $A \cdot x + d$. Dadurch ist $J(A \cdot x + d) = I(x)$. Ein Feature zu tracken bedeutet also die vier Parameter der Transformationsmatrix und die zwei Parameter des Translationsvektors zu bestimmen.

Einen einzelnen Bildpunkt zu tracken ist nicht möglich (siehe Kapitel 1.2). Deswegen muss immer die Nachbarschaft eines Feature Bildpunkts miteinbezogen werden. Bildpunkte der Nachbarschaft, die zum selben Objekt der Wirklichkeit gehören, vollziehen fast die gleiche affine Bewegung.

Der KLT Tracker verwendet für die Nachbarschaft immer ein Fenster mit fixer Seitenlänge. Die Qualität der Berechnung hängt von der Fenstergröße ab, die man für das Feature zu Grunde legt, von der Texturiertheit des Fensterinhalts und von der Kamerabewegung [27]. Wenn das Fenster klein ist, ist es schwierig, die Parameter der Transformationsmatrix zu bestimmen. Bei zu groß gewählten Fenstern treten andererseits Tiefenunstetigkeiten an Objektgrenzen häufiger auf, da ein großes Fenster nicht mehr nur einen Ausschnitt von einem Objekt zeigt, sondern über die Objektgrenze hinaus auch Teile von anderen Objekten.

Bei einem Fenster, in dem Objektgrenzen vorhanden sind, ist die Berechnung der affinen Parameter ungenau. Es werden nämlich Bildpunkte zur Berechnung herangezogen, die nicht zu dem untersuchten Objekt gehören – und damit einer anderen Bewegung zugrunde liegen. Beim Region Growing hingegen wird die Nachbarschaft daraufhin getestet, ob sie die affinen Parameter der Region unterstützen. Nach jedem Wachstumsschritt, der die Region vergrößert, werden die affinen Parameter der Region neu berechnet. Je größer die Region wird, desto genauer sind die affinen Parameter bestimmt. Durch das selektive Einbeziehen von Nachbarschaftsbildpunkten wird vermieden, dass Bildpunkte für die Berechnung der affinen Bewegung herangezogen werden, die in Wirklichkeit eine ganz andere Bewegung vollziehen.

In beiden Fällen besteht das Bestimmen der affinen Parameter im Minimieren der Funktion (2.4).

$$\epsilon = \int \int_W [J(Ax + d) - I(x)]^2 \, w(x) dx \tag{2.4}$$

Dabei ist $W$ die Menge der Bildpunkte, für die die affine Bewegung berechnet wird. Beim KLT Tracker sind das die Bildpunkte des Feature Fensters. Beim Region Growing sind das die Bildpunkte der Region. $w(x)$ ist eine Gewichtsfunktion, welche die im Zentrum der Region liegenden Punkte stärker berücksichtigt als Punkte in größerem Abstand vom Zentrum. In dieser Arbeit werden alle Bildpunkte der Region gleich gewichtet, darum ist $w(x) = 1$. Die Funktion bestimmt die Summe der quadratischen Differenzen jedes Bildpunkts der Region. Die Differenz ist die Intensitätsdifferenz eines Bildpunkts in Bild $I$ und dem affin abgebildeten Bildpunkt in Bild $J$.

Um die Funktion zu minimieren, wird sie im ersten Schritt differenziert und das Ergebnis auf 0 gesetzt. Die erste Ableitung einer Funktion entspricht ihrer Steigung. Wenn die Steigung 0 ist, ist das ein Minimum oder ein Maximum der Funktion. Da es sich bei der ersten Ableitung der Gleichung (2.4) aber um eine nichtlineare Funktion handelt, kann die Nullstelle nicht direkt ermittelt werden. Darum wird das Newtonsche Näherungsverfahren verwendet, das sich für stetig differenzierbare Funktionen eignet. Dabei wird die nichtlineare Funktion an einem Startpunkt linearisiert, also die Tangente an diesem Punkt berechnet. Die Nullstelle der Tangente bildet dann den neuen Startpunkt, an dem wieder die Tangente gebildet und die neue Nullstelle bestimmt wird. Wiederholt man diese Schritte, wird der Fehler immer kleiner und eine genäherte Nullstelle

der nichtlinearen Funktion gefunden. Durch die Taylorentwicklung wird die nichtlineare Funktion an einem bestimmten Punkt liniarisiert, was zu der Gleichung (2.5) führt. Das Verfahren wird in [26] genau beschrieben.

$$Tz = a \tag{2.5}$$

Dabei enthält der Vektor $z = [d_{xx}\, d_{xy}\, d_{yx}\, d_{yy}\, d_x\, d_y]$ die genäherten Parameter der Transformationsmatrix und des Translationsvektors. $a$ ist der Fehlervektor und $T$ ist eine $6x6$ Matrix, die durch ein Bild bestimmt werden kann. Die Berechnung von $T$ wird in Gleichung (2.6) dargestellt.

$$T = \iint_W \begin{bmatrix} x^2g_x^2 & x^2g_xg_y & xyg_x^2 & xyg_xg_y & xg_x^2 & xg_xg_y \\ x^2g_xg_y & x^2g_y^2 & xyg_xg_y & xyg_y^2 & xg_xg_y & xg_y^2 \\ xyg_x^2 & xyg_xg_y & y^2g_x^2 & y^2g_xg_y & yg_x^2 & yg_xg_y \\ xyg_xg_y & xyg_y^2 & y^2g_xg_y & y^2g_y^2 & yg_xg_y & yg_y^2 \\ xg_x^2 & xg_xg_y & yg_x^2 & yg_xg_y & g_x^2 & g_xg_y \\ xg_xg_y & xg_y^2 & yg_xg_y & yg_y^2 & g_xg_y & g_y^2 \end{bmatrix} w\, dx \tag{2.6}$$

Dabei sind $x$ und $y$ die Koordinaten eines Bildpunkts der Region $W$. $g_x$ und $g_y$ sind die Werte des Gradienten des Bildes $J$ in $x$ und $y$ Richtung an der Stelle des Bildpunkts der Region. Die Berechnung von $a$ ist in Gleichung (2.7) dargestellt.

$$a = \iint_W [I(x) - J(x)] \begin{bmatrix} xg_x \\ xg_y \\ yg_x \\ yg_y \\ g_x \\ g_y \end{bmatrix} w\, dx \tag{2.7}$$

Eine Bestimmung der unbekannten Parameter von $z$ führt zu einer ersten Näherung des Ergebnisses. Mit Hilfe des Newton Verfahrens kann man den Wert des Fehlervektors $a$ Schritt für Schritt bis zu einem festgelegten Wert verkleinern. Der Vektor $z$ enthält dann die genäherten Parameter der Transformationsmatrix und des Translationsvektors.

### 2.2.2 Gute Features

Nicht jeder Punkt in einem Bild eignet sich als Feature. Beispielsweise kann man ein Feature, das auf einer Linie[1] liegt, nur in eine Richtung verfolgen. Denn entlang der Linie unterscheiden sich die Punkte nicht, und jeder würde als Feature infrage kommen – dieses Problem wird als *Aperture Problem* bezeichnet. Eine Ecke hingegen kann in alle Richtungen verfolgt werden. In der Literatur werden Punkte vorgeschlagen, die eine hohe Ortsfrequenz aufweisen oder bei denen die zweite Ableitung der Region hoch ist [27].

[1] Jeder Punkt auf einer Linie besitzt die gleiche Intensität und unterscheidet sich dadurch nicht von den anderen Punkten auf der Linie.

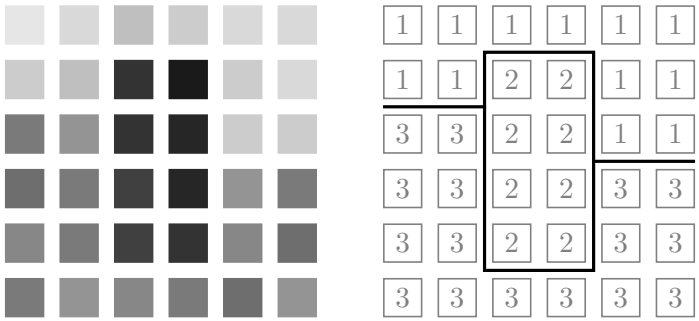



Abbildung 2.2: Beispiel für eine Zuordnung von Label zu einzelnen Bildpunkten eines Bildes.

Der KLT Tracker benutzt eine eigene Definition von guten Features. Ein gutes Feature ist demnach ein Feature, das leicht getrackt werden kann, das also den Tracking Erfolg erhöht. Das bedeutet, dass das Tracking Kriterium – die Gleichung, deren Lösung die affinen Parameter bestimmt – günstig aufgestellt sein soll. Ein Feature, das die Parameter der Gleichung günstig festlegt, ist ein gutes Feature.

## 2.3 Zuordnungsaufgaben in Computer Vision

Viele Aufgaben in Computer Vision bestehen darin, jedem Bildpunkt einen *Label* aus einer Menge möglicher Label zuzuweisen. Einige Teilgebiete von Computer Vision sind Stereo Vision, Motion, Bildrestaurierung und Bildsegmentierung. Stereo Vision beschäftigt sich mit dem räumlichen Sehen mit zwei Kameras. Die Label entsprechen hier Disparitäten. Bei Motion entsprechen die Label Bewegungsinformationen, dabei handelt es sich meistens um affine Bewegungen, also Ebenenbewegungen. Die Bildrestaurierung beschäftigt sich mit dem Entfernen von Rauschen in Bildern. Die Label entsprechen dabei einer Intensität oder Farbe. Bei der Bildsegmentierung entsprechen die Label einem Objekt oder Gegenstand. Abbildung 2.2 zeigt, wie das Ergebnis einer Zuordnung aussehen könnte, welche Bildpunkten unterschiedlicher Intensität drei Label zuordnet. Das Beispiel ist abstrakt und soll nur die Zuordnungsaufgabe veranschaulichen. Man kann sich aber vorstellen, dass es sich um ein Beispiel aus der Bildrestaurierung handelt. In diesem Fall entsprechen die Label $\{1, 2, 3\}$ einer Farbe oder einem Grauwert.

Bei all diesen Zuordnungsaufgaben handelt es sich um die gleiche Problemklasse. Bei der Zuordnung der Label zu den einzelnen Bildpunkten gibt es viele Lösungen – eine zufällige Zuordnung ist beispielsweise auch eine Lösung. In Computer Vision wird meistens eine Funktion definiert, welche eine Lösung bewertet. Diese Funktion besteht gewöhnlich aus mehreren Termen. Ein Term davon bildet die Daten ab, ein anderer Teil bildet das Erfahrungswissen ab. Der Datenterm ist nämlich nicht ausreichend, weil Sensorrauschen, Objektivfehler und andere Einflüsse die theoretische Farbe und Intensität eines Bildpunkts beeinflussen.

Eine Funktion, welche eine Zuordnungslösung bewertet, wird als *Energiefunktion* be-

zeichnet. Die Energiefunktion liefert für eine *Zuordnung* $f$ einen Zahlenwert, der die Qualität der Lösung repräsentiert. Je kleiner dieser Zahlenwert ist, desto besser ist die Lösung. Die Herausforderung besteht darin, eine passende Energiefunktion für ein bestimmtes Problem zu entwerfen und zum anderen im Suchen einer Zuordnung, welche ein Minimum der Energiefunktion liefert. Da in der Energiefunktion meistens ein Smoothnessterm enthalten ist, der zwei Variablen – die Zuordnung des untersuchten Bildpunkts und die Zuordnung eines Nachbarbildpunkts – berücksichtigt, ist das Auffinden des Minimums schwierig. Außerdem sind die interessantesten Energiefunktionen nicht konvex, das heißt sie haben neben dem globalen Minimum mindestens ein lokales Minimum. Bereits die einfachste Form der Discontinuity Preserving Energiefunktionen, das Potts Modell, ist bewiesenermaßen NP vollständig [22]. Aus diesen Gründen wird die Energiefunktion oft angepasst, damit sich das Suchen nach dem Minimum leichter gestaltet [31].

Weil das Minimieren vieler interessanter Energiefunktionen NP schwer ist, wird nach Näherungsverfahren gesucht, die eine gute Lösung finden. Lange Zeit wurden Energiefunktionen dieser Art durch iterative Algorithmen, wie *Simulated Annealing*, gelöst. Dieses Verfahren ist aber sehr langsam. Eines der am viel versprechendsten ist das Verfahren des *minimalen Schnitts – Graph Cut*, in der englischsprachigen Literatur –, das zuerst von Greig et al. vorgeschlagen wurde [12]. Dieses hat lange Zeit wenig Beachtung gefunden, weil es nur zwei Label zuordnen konnte [6]. Boykov et al. haben in ihrer Arbeit [10] gezeigt, wie man das Verfahren des minimalen Schnitts mit dem Expansion Move bzw. dem $\alpha\beta$ Swap Move verwenden kann, um Energiefunktionen mit mehreren Label zu lösen. Kolmogorov und Zabih haben in ihrer Arbeit [22] ein Verfahren beschrieben, wie man eine Energiefunktion mit Termen, die bis zu drei Variable beinhalten können, in einem Flussnetzwerk für das Expansion Move Verfahren abbilden kann.

Im Folgenden wird auf die Energiefunktion genauer eingegangen. Die typische Form einer Energiefunktion ist in Gleichung (2.8) dargestellt.

$$E(f) = E_{smooth}(f) + E_{data}(f) \tag{2.8}$$

$$E_{data}(f) = \sum_{p \in P} D_p(f_p) \tag{2.9}$$

$$E_{smooth}(f) = \sum_{\{p,q\} \in N} V_{\{p,q\}}(f_p, f_q) \tag{2.10}$$

Dabei ist $E_{data}$ die Summe der Datenkosten jedes Bildpunkts. Datenkosten sind Kosten, die entstehen, wenn einem Bildpunkt ein Label zugeordnet wird. Dabei sind die Kosten größer, wenn der Label schlecht zu ihm passt. Sie sind geringer, wenn er gut zu ihm passt. Angenommen man möchte ein Graustufenbild in ein Schwarzweißbild umwandeln. In der Funktion sollen dann große Kosten entstehen, wenn einem relativ hellen Bildpunkt der Label Schwarz zugeordnet wird, und es sollen kleine Kosten entstehen, wenn dem Bildpunkt der Label Weiß zugeordnet wird.

$D_p$ drückt die Kosten aus, die entstehen, wenn einem Bildpunkt ein gewisser Label $f_p$ zugeordnet wird. Die Datenkostenfunktion könnte zum Beispiel $D_p = |I(f_p) - i_p|$ lauten. $i_p$ drückt dabei die Intensität oder Farbe des Bildpunkts $p$ aus. $I(f_p)$ drückt

dabei die Intensität oder Farbe des Labels aus, der dem Bildpunkt $p$ in der Zuordnung $f$ zugewiesen wird.

$E_{smooth}$ ist die Summe der Smoothnesskosten jedes benachbarten Bildpunktpaares. Smoothnesskosten entstehen, wenn zwei benachbarte Bildpunkte unterschiedliche Label bekommen. Die Smoothnesskosten zwischen benachbarten Bildpunkten mit ähnlichen Eigenschaften sollen größer sein als die Smoothnesskosten zwischen benachbarten Bildpunkten mit unterschiedlichen Eigenschaften. Angenommen im Graustufenbild ist eine Region von hellen Bildpunkten. Unter diesen Bildpunkten ist ein Bildpunkt, der für den Label Weiß gerade nicht hell genug ist und vom Datenterm deshalb den Label Schwarz zugeordnet bekäme. Hier wirken die Smoothnesskosten dagegen. Denn wenn dieser eine Bildpunkt von seinen vier Nachbarn getrennt würde, dann entstünden viermal Smoothnesskosten. Wenn dieser Bildpunkt den Label Schwarz bekäme, dann müsste die Summe dieser vier Smoothnesskosten kleiner sein als die Datenkosten für den Label Schwarz. Während die Datenkosten genau das repräsentieren, was beobachtet wird, repräsentieren die Smoothnesskosten das Wissen über den Sachbereich (Motion, Restaurierung, Stereo Vision, ...). Diese Kosten sind sinnvoll, da es viele Gründe – siehe Kapitel 1.2 – gibt, warum ein einzelner Bildpunkt bezüglich seiner Eigenschaften (Farbe, Helligkeit) nicht für sich alleine betrachtet werden kann.

$V_{\{p,q\}}$ drückt die Smoothnesskosten aus. Die Smoothnesskostenfunktion könnte zum Beispiel $V_{\{p,q\}} = |f_p - f_q|$ lauten, in diesem speziellen Fall kann die Energiefunktion sogar global optimal gelöst werden [10, 9].

Für ein binäres Zuordnungsproblem – wenn die Aufgabe darin besteht, jedem Bildpunkt einen von zwei möglichen Label zuzuordnen – gibt es global optimale Verfahren. Eines ist das Verfahren des *minimalen Schnitts* [8, 10], das in dieser Arbeit in 2.4 vorgestellt wird.

Ein Zuordnungsproblem, in dem mehr als zwei Label zuzuordnen sind und bei dem die Energiefunktion mehrere lokale Minima besitzt, ist NP vollständig [22]. Ein exaktes Lösen der Minimierungsaufgabe ist mit einem vertretbaren Aufwand dann nicht möglich. Es gibt aber Näherungsverfahren, die sehr gute Ergebnisse erzielen. In dieser Arbeit werden das *Expansion Move* und $\alpha\beta$ *Swap* Verfahren in 2.5 vorgestellt, bei denen das Verfahren des minimalen Schnitts ein Hauptbestandteil ist.

## 2.4 Minimaler Schnitt - Maximaler Fluss

Das Bestimmen des minimalen Schnitts und des maximalen Flusses sind äquivalente Probleme. Das heißt, wenn der maximale Fluss bestimmt ist, ist auch der minimale Schnitt bestimmt und umgekehrt. Das Maximaler-Fluss-Minimaler-Schnitt-Theorem besagt, dass der Wert des maximalen Flusses gleich der Kapazität eines minimalen Schnitts ist. Ein minimaler Schnitt wird an einem Flussnetzwerk durchgeführt. Ein Flussnetzwerk ist ein gerichteter Graph mit zwei speziellen Knoten, der Quelle $s$ und der Senke $t$. Der Maximale Fluss ist dann der größtmögliche Fluss, den man von der Quelle zur Senke schicken kann, ohne dass Kapazitätsbeschränkungen überschritten werden [11]. Im Folgenden werden der minimale Schnitt und andere verwendete Begriffe an einem Beispiel

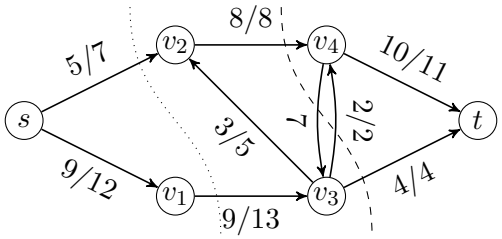



Abbildung 2.3: Zwei Schnitte in einem Flussnetzwerk. Der minimale Schnitt ist durch die gestrichelte Linie dargestellt. Die Zahlen an den Kanten zeigen den aktuellen Fluss und die Kapazität der Kante an. Wenn der Fluss 0 ist, dann ist nur die Kapazität der Kante vermerkt.

erklärt. Abbildung 2.3 zeigt zwei Schnitte in einem Flussnetzwerk mit einer Quelle $s$ und einer Senke $t$. Der Wert eines Flusses in einem Flussnetzwerk ist durch Gleichung (2.11) definiert. Er ist also gleich dem gesamten Fluss aus der Quelle heraus. Der Wert des Flusses in Abbildung 2.3 ist demzufolge $|f| = 14$.

$$|f| = \sum_{v \in V} f(s, v) \tag{2.11}$$

Der minimale Schnitt wird nach Cormen, Leiserson, Rivest und Stein [11] wie folgt definiert: "Die Kapazität eines Schnitts $(S, T)$ ist $c(S, T)$. Ein minimaler Schnitt eines Flussnetzwerks ist ein Schnitt, dessen Kapazität die kleinste von allen Schnitten des Netzwerks ist". Der minimale Schnitt teilt den Graphen in diesem Beispiel in die Mengen $S = \{s, v_1, v_2, v_3\}$ und $T = \{t, v_4\}$. In die Berechnung der Kapazität des Schnitts $c(S, T)$ gehen nur Kanten ein, die von $S$ nach $T$ verlaufen. Die Kapazität des minimalen Schnitts in diesem Beispiel ist $c(\{s, v_1, v_2, v_3\}, \{t, v_4\}) = 14$, es gibt keinen Schnitt, der eine kleinere Kapazität hat. Die Kapazität des Schnitts, der durch die punktierte Kurve dargestellt ist, beträgt 20, ist also größer.

Der entscheidende Schritt im Verfahren ist das Suchen nach Erweiterungspfaden – das ist ein Pfad von der Quelle zur Senke, über Kanten, die noch zusätzlichen Fluss aufnehmen können, deren Kapazitätsbeschränkung also noch nicht erreicht ist. Im Wesentlichen gibt es zwei Gruppen von Algorithmen, mit denen ein Erweiterungspfad gesucht werden kann. Die eine Art kann man der *push relabel* Gruppe und die andere Art der *augmenting paths* Gruppe zuordnen.

Die Verfahren werden allgemein in [11] beschrieben. Der Algorithmus, der in dieser Arbeit verwendet wird, wurde von Vladimir Kolmogorov und Yuri Boykov entwickelt und gehört zur *augmenting paths* Gruppe. Beschrieben ist der Algorithmus in der Arbeit [6].

## 2.5 $\alpha$-Expansion und $\alpha\beta$-Swap Verfahren

Expansion Move und $\alpha\beta$ Swap Move sind iterative Verfahren, die in jedem Iterationsschritt einen Label zuordnen. Der Expansion Move sollte gegenüber dem $\alpha\beta$ Swap Move

(a) Orginal

(b) Expansion Move

(c) $\alpha\beta$ Swap Move

Abbildung 2.4: Ein Schritt im Expansion Move Algorithmus (b) und ein Schritt im $\alpha\beta$ Swap Algorithmus in (c).

bevorzugt werden, da er eine bessere Laufzeit hat. Ein weiterer Vorteil ist, dass sich bestimmen lässt, wie weit ein lokales Minimum, im schlimmsten Fall, vom globalen Minimum der Energiefunktion entfernt ist.

Da der Expansion Move Algorithmus in einem Durchlauf jeden Label nur einmal prüfen muss, hat er eine Laufzeit von $O(n)$. Der $\alpha\beta$ Swap Move muss in einem Durchlauf jeden Label mit jedem anderen Label prüfen, dadurch hat er eine Laufzeit von $O(n!)$. Abbildung 2.4 zeigt, dass beim $\alpha\beta$ Swap Algorithmus in einem Schritt (Move) nur Bildpunkte mit einem Label $\alpha$ gegen einen Label $\beta$ ersetzt werden können und umgekehrt. Beim Expansion Move Algorithmus dagegen kann in einem Schritt jeder Bildpunkt, egal welchen Label er hat, den Label $\alpha$ erhalten. Das ist ein entscheidender Effizienzvorteil. Allerdings unterliegt der Expansion Move Algorithmus der Einschränkung, dass er nur bei metrischen Smoothfunktionen anwendbar ist, das heißt, die Dreiecksungleichung $V(\alpha, \beta) \leq V(\alpha, \gamma) + V(\gamma, \beta)$ muss erfüllt sein. Für beide Verfahren muss $V(\alpha, \beta) = V(\beta, \alpha)$ gelten [10]. Der $\alpha\beta$ Swap Move Algorithmus kann somit für eine größere Gruppe von Energiefunktionen angewandt werden, weil er auch für semimetrische Smoothfunktionen gilt. Die besseren Ergebnisse des Expansion Move Algorithmus wurden experimentell in [8, 28] festgestellt. In [8] wird auch ein Beweis dafür gegeben, wie weit das lokale Optimum maximal vom globalen Optimum entfernt sein kann.

Sowohl $\alpha\beta$ als auch der Expansion Move Algorithmus sind in der Arbeit [10] beschrieben. In dieser Arbeit wird der Expansion Move Algorithmus verwendet, auf den im Folgenden genauer eingegangen wird.

Abbildung 2.5 stellt den Expansion Move Algorithmus dar. Er beginnt mit einer willkürlichen Zuordnung der Label zu den Bildpunkten. Für jeden Label wird der Expansion Move durchgeführt. Es gilt, den Expansion Move zu finden, der die besten Kosten unter allen möglichen Expansion Moves hat. Wenn der Expansion Move die Energie verbessert, dann wird die Zuordnung durch diesen Move aktualisiert. Wenn wenigstens ein Move die Energie verbessert hat, dann wird wieder von vorne begonnen und wieder für jeden La-

**Input** : Videosequenz, Menge affiner Parameter $A$
**Output** : Die Zuordnung zu den Bildpunkten $f$

```
1  Beginne mit einer willkürlichen Zuordnung f;
2  repeat
3  |  fertig ← true;
4  |  foreach Label α ∈ A do
5  |  |  f̂ ← Expansion Move von α;
6  |  |  if E(f̂) < E(f) then                    /* Sind Kosten geringer? */
7  |  |  |  fertig ← false;
8  |  |  |  f ← f̂;
9  |  |  end
10 |  end
11 until fertig ;
```

Abbildung 2.5: Expansion Move Algorithmus

bel der Expansion Move durchgeführt. Der entscheidende Schritt im Algorithmus ist in Zeile 5 dargestellt, dieser wird im Folgenden genauer erklärt.

Für den Expansion Move gibt es zwei Algorithmen, die sich darin unterscheiden, wie das Flussnetzwerk für den minimalen Schnitt erstellt wird. Der erste Algorithmus ist von Yuri Boykov, Olga Veksler und Ramin Zabih [10], der zweite Algorithmus ist von Vladimir Kolmogorov und Ramin Zabih [22] entwickelt worden. Der erste Algorithmus hat einen Effizienznachteil, weil das Flussnetzwerk mehr Knoten und Kanten benötigt als der zweite Algorithmus. Die Qualität der Ergebnisse ist aber dieselbe, und weil der erste einfacher zu verstehen ist, wird er in dieser Arbeit verwendet und vorgestellt.

Die Quelle im Expansion Move Algorithmus entspricht dem Label $\alpha$ und die Senke dem Label $\overline{\alpha}$ – also dem Label, den ein Bildpunkt im Augenblick besitzt. Für jeden Bildpunkt wird ein Knoten in das Flussnetzwerk eingefügt (siehe Abbildungen 2.6 und 2.7). Von jedem Knoten wird eine Kante zur Quelle eingetragen, die das Gewicht erhält, das den Datenkosten $D_p(\alpha)$ entspricht, wenn der Bildpunkt den Label $\alpha$ erhält. Von jedem Knoten wird eine Kante zur Senke eingetragen, die das Gewicht erhält, das den Datenkosten $D_p(f_p)$ entspricht, wenn der Bildpunkt seinen derzeitigen Label $f_p$ erhält. Für benachbarte Bildpunkte $p$ und $q$ müssen Smoothnesskanten eingetragen werden. Dabei müssen zwei Fälle unterschieden werden.

Haben $p$ und $q$ den gleichen Label in $f$, also $f_p = f_q$, dann wird eine Smoothnesskante mit den Smoothnesskosten $V_{\{p,q\}}(\alpha, f_p)$ zwischen $p$ und $q$ eingetragen. Das Flussnetzwerk, für benachbarte Bildpunkte mit gleichem Label, ist in Abbildung 2.6 dargestellt. Haben aber $p$ und $q$ unterschiedliche Label in $f$, also $f_p \neq f_q$, dann gibt es zwischen den Bildpunkten $p$ und $q$ insgesamt drei Möglichkeiten, wie Smoothnesskosten entstehen können.

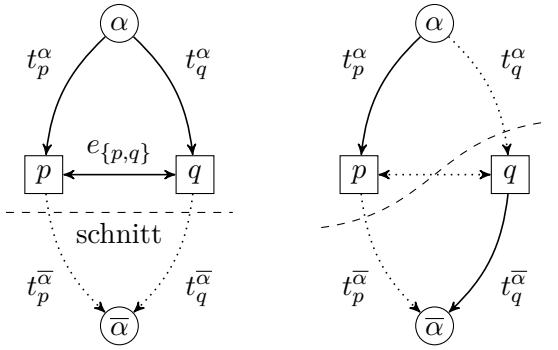
1. Wenn $p$ den alten Label behält und $q$ den Label $\alpha$ erhält, sind die Kosten $V(f_p, \alpha)$.

2. Wenn $p$ den Label $\alpha$ erhält und $q$ seinen alten Label behält, sind die Kosten $V(\alpha, f_q)$.

3. Wenn $p$ und $q$ ihre alten Label behalten, sind die Kosten $V(f_p, f_q)$.

Durch Hinzufügen eines zusätzlichen Knotens $a$ zwischen $p$ und $q$ können alle drei Smoothnesskosten in das Flussnetzwerk eingetragen werden. Abbildung 2.7 veranschaulicht diese Konstruktion. Die Kante $e_p$ erhält das Gewicht $V_{\{p,q\}}(f_p, \alpha)$, die Kante $e_q$ erhält das Gewicht $V_{\{p,q\}}(\alpha, f_q)$, und die Kante zur Senke $e_a$ erhält das Gewicht $V_{\{p,q\}}(f_p, f_q)$. In der Abbildung ist auch ersichtlich, warum die Dreiecksungleichung $V(\alpha, \beta) \leq V(\alpha, \gamma) + V(\gamma, \beta)$ gelten muss, denn es darf nur eine Smoothnesskante bei benachbarten Bildpunkten durchtrennt werden. Durch die Minimaleigenschaft des minimalen Schnitts und der Dreiecksungleichung ist diese Bedingung erfüllt. Der Schnitt durch die Kanten $t_p^{\alpha}$, $e_p$, $e_a$ und $t_q^{\overline{\alpha}}$ ist zum Beispiel nicht möglich, da $e_q \leq e_p + e_a$ ist.

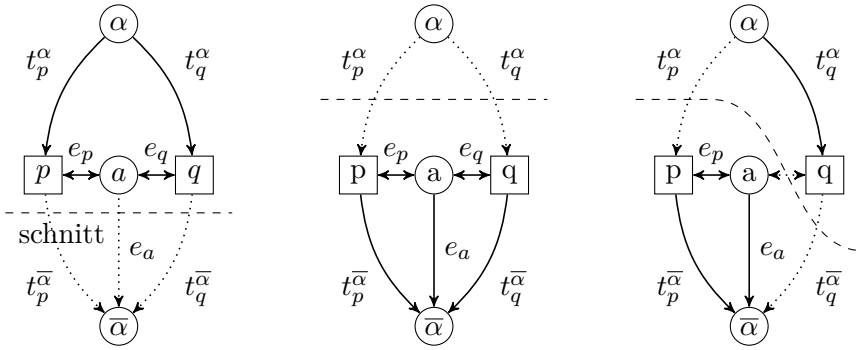
Der minimale Schnitt teilt das Flussnetzwerk in zwei Knotenmengen. Jene Knoten, die von der Quelle getrennt wurden, erhalten den der Quelle zugeordneten Label $\alpha$. Jene Knoten, die von der Senke getrennt wurden, erhalten den der Senke zugeordneten Label $\overline{\alpha}$ – der Bildpunkt behält damit seinen alten Label.




(a) $p$ und $q$ behalten ihren ursprünglichen Label

(b) $p$ behält seinen ursprünglichen Label und $q$ bekommt den Label $\alpha$

Abbildung 2.6: Zwei von vier möglichen Schnitten durch ein Flussnetzwerk mit zwei benachbarten Bildpunkten $p$ und $q$. Die $t$ Kanten entsprechen Datenkosten, die $e$ Kanten entsprechen Smoothnesskosten ($e_{\{p,q\}} = V_{\{p,q\}}(\alpha, \overline{\alpha})$). Die Summe der Kantengewichte entspricht der Kapazität des Schnitts, bzw. dem Maximalen Fluss.




(a) $p$ und $q$ behalten ihren ursprünglichen Label

(b) $p$ und $q$ erhalten den Label $\alpha$

(c) $p$ erhält den Label $\alpha$ und $q$ behält seinen ursprünglichen Label

Abbildung 2.7: Drei von vier möglichen Schnitten durch einen Graphen von zwei benachbarten Bildpunkten $p$ und $q$, die unterschiedliche Label haben ($f_p \neq f_q$). Die $t$ Kanten entsprechen Datenkosten, die $e$ Kanten entsprechen Smoothnesskosten. Die Summe der Gewichte der gepunkteten Kanten entspricht der Kapazität des Schnitts, bzw. dem Maximalen Fluss.

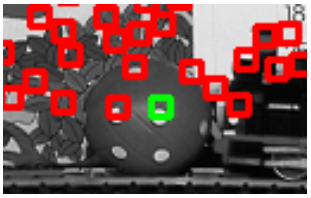
# 3 Algorithmus

In dieser Arbeit wird ein Ansatz gewählt, wie ihn J. Xiao und M. Shah in ihren Arbeiten [35, 36] verfolgt haben. Die Ergebnisse, die sie mit ihrem Ansatz erzielen, sind vergleichbar mit anderen modernen Algorithmen in diesem Gebiet. Ihr Algorithmus verwendet zum ersten Mal explizit Verdeckungen in Motion Estimation [35]. In Stereo Vision hat die Berücksichtigung von Verdeckungen schon früher zu einer Verbesserung der Ergebnisse geführt. In Motion Estimation wurden sie aber bislang nicht betrachtet. Aus diesen Gründen wird in dieser Arbeit versucht, den Algorithmus aus [35, 36] nachzubauen. Die anfangs beschriebene Aufgabe, jedem Bildpunkt eine Bewegung zuzuweisen, lässt sich in zwei Teilaufgaben trennen.

Die erste Teilaufgabe besteht darin festzustellen, welche affinen Bewegungen in einem Video vorkommen. Diese Menge soll vollständig sein und untereinander verschiedene affine Bewegungen beinhalten. Dieser Schritt wird auch als *Layer Extraction Schritt* bezeichnet und wird im Kapitel 3.2 dargestellt. Der Ablauf dieses Schritts stammt aus der Arbeit [35], welcher dort gut beschrieben wird. Der Schritt beginnt mit dem identifizieren von Featurepoints. In allen Bildern der Sequenz wird versucht, diese wiederzufinden und so ihren Weg zu *Tracken*. Diese getrackten Features sind der Ausgangspunkt, um die affinen Bewegungen zu erhalten. Ein Region Growing Algorithmus benutzt jeden Featurepoint als Saatpunkt, um ihn zu einer Region wachsen zu lassen. Dabei wird die affine Bewegung in jeder Wachstumsphase immer genauer bestimmt. Features, die demselben Objekt in der Wirklichkeit angehören, teilen sich dieselben affinen Parameter. Diese, einander sehr ähnlichen, affinen Bewegungen werden durch einen Region Merging Algorithmus erkannt und zu einer affinen Bewegung zusammengefasst. Das Ergebnis ist eine Menge affiner Bewegungen für die zweite Teilaufgabe.

Diese besteht darin, jedem Bildpunkt des Videos die am besten passende affine Bewegung aus dieser Menge zuzuordnen. Dieser Schritt wird auch als *Layer Assignment Schritt* bezeichnet und ist im Kapitel 3.3 dargestellt. Die Energiefunktion für diesen Schritt stammt aus der Arbeit [36]. Nachdem in [36] Details für die Berechnung der Smoothnesskosten und Datenkosten fehlen, wurde versucht diese so zu Berechnen, wie es in [35] beschrieben wird. Entwurfdetails, die in den Orginalarbeiten nicht beschrieben wurden, sind durch eigene Überlegungen ergänzt worden – siehe Kapitel 3.3.3. Da es sich bei diesem Schritt um ein NP vollständiges Problem handelt, wird das Expansion Move Näherungsverfahren verwendet.

Im Folgenden werden die zwei Teilaufgaben ausführlich behandelt. Davor soll ein einführendes Kapitel den Algorithmus beispielhaft erklären.



(a) Gute Features im Referenzbild

(b) Wiedergefundene Features

Abbildung 3.1: Bild (a) sind die vom KLT Tracker ausgewählten Features im Referenzbild. Die Features werden Bild für Bild getrackt. Bild (b) zeigt die gefundenen Features im letzten Bild. Der Ball rollt – von der Eisenbahn angeschoben – von rechts nach links. In Bild (b) ist er gegenüber Bild (a) gegen den Uhrzeigersinn rotiert.

## 3.1 Einführendes Beispiel

In diesem Beispiel wird die Arbeitsweise des Algorithmus allgemein dargestellt. Das Ziel dieses einführenden Beispiels ist, einen Überblick zu geben. Dadurch lassen sich die Algorithmendetails, die später in den entsprechenden Kapiteln erläutert werden, leichter in den Gesamtkontext einordnen. Der Algorithmus wird in diesem Beispiel mit der *Mobile & Calendar* Sequenz untersucht. In dieser Sequenz fährt eine Modelleisenbahn, die einen Ball vor sich herrollt. Die Tapete im Hintergrund und die Schienen sind starr. Der Kalender bewegt sich nach unten. Die Eisenbahn fährt nach links. Neben der Objektbewegung gibt es noch eine Kamerabewegung, die dem Zug nach links folgt. Außerdem ändert sich der Bildinhalt durch einen leichten zoom-out der Kamera. Im Folgenden werden die Ergebnisse gezeigt, die den Ball in der Sequenz betreffen.

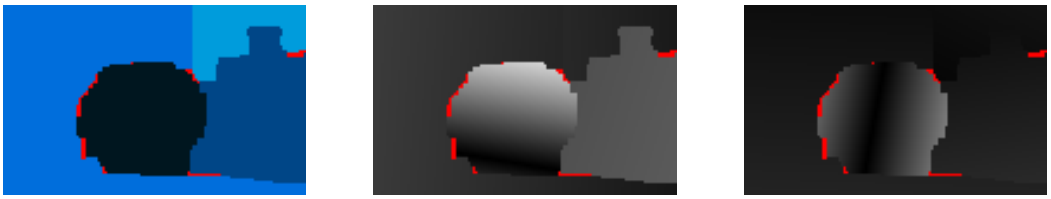
Der Layer Extraction Schritt beginnt mit dem Suchen von Featurepoints, welche über alle Bilder der Eingabesequenz getrackt werden können. Das Suchen der Featurepoints wird vom KLT Tracker [15, 27] realisiert. Gesucht werden die Featurepoints im ersten Bild. Jeder Featurepoint wird von Bild zu Bild getrackt. Kann ein Featurepoint in einem Bild nicht mehr gefunden werden, dann ist er nicht verwendbar und wird ausgeschieden. Darum soll die Eingabesequenz nur wenige Bilder umfassen, damit nicht zu viele Featurepoints beim Tracken verlorengehen. In dieser Arbeit hat die Eingabesequenz meistens fünf Bilder. In Abbildung 3.1 sieht man die über alle Bilder der Eingabesequenz getrackten Features. Dargestellt ist das erste und das letzte Bild der Sequenz. Im Folgenden wird der Featurepoint untersucht, der in der Abbildung grün dargestellt ist.

Der KLT Tracker berechnet für jeden Featurepoint die affinen Parameter. Diese beziehen sich aber auf ein Fenster mit fixer Seitenlänge und sind deshalb nicht repräsentativ für das Objekt – in diesem Fall der Ball. Damit der Layer Assignment Schritt gute Ergebnisse liefern kann, benötigt er repräsentative affine Parameter. Aus diesem Grund wird Region Growing eingesetzt, um die affinen Parameter exakter festzustellen.

Region Growing beginnt mit einem Saatpunkt und überprüft in einem iterativen Verfahren die Nachbarschaft. Die Region wächst solange, bis keine Nachbarschaft mehr da

Tabelle 3.1: Die Affinen Parameter des Ball Layers im Vergleich, vor dem Region Growing und danach. Der Translationsvektor kann durch die Koordinaten bestimmt werden, $t = (x_2 - x_1, y_2 - y_1)$. Das Bild in der ersten Zeile und rechten Spalte der Abbildung 3.3 entspricht den affinen Parametern in der ersten Zeile. Das Bild in der letzten Zeile und rechten Spalte entspricht den affinen Parametern der zweiten Zeile.

| | $x_1$ | $y_1$ | $x_2$ | $y_2$ | $a_{xx}$ | $a_{xy}$ | $a_{yx}$ | $a_{yy}$ |
|---|---|---|---|---|---|---|---|---|
| Vorher | $166,0$ | $203,0$ | $161,3$ | $201,3$ | $1,005$ | $-0,110$ | $0,150$ | $0,984$ |
| Nachher | $166,0$ | $203,0$ | $161,2$ | $201,2$ | $0,966$ | $-0,167$ | $0,200$ | $0,970$ |



(a) Fertige Zuordnung

(b) X Komponente der Bewegungsvektoren

(c) Y Komponente der Bewegungsvektoren

Abbildung 3.2: Das Ergebnis des Layer Assignment Schritts. Die roten Bereiche kennzeichnen verdeckte Bildpunkte. Helle Bereiche in Bild (b) und (c) weisen auf Bewegung hin, schwarze Bereiche auf keine Bewegung.

ist, welche zur Region gezählt werden kann. Das Verfahren bewertet die Nachbarschaft unterschiedlich. Bildpunkte, die nahe der Region sind, haben eine größere Chance, in die Region aufgenommen zu werden. Nach jeder Wachstumsphase werden die affinen Parameter der Region neu berechnet. Abbildung 3.3 zeigt das Verfahren für den grünen Featurepoint aus Abbildung 3.1. Für den Ball Layer ergeben sich nach dem Region Growing die affinen Parameter aus Tabelle 3.1.

Nach dem Region Growing sind viele affine Parameter mehrmals bestimmt worden, da für jeden Featurepoint das Region Growing durchgeführt wurde. Im Region Merging Schritt werden sehr ähnliche affine Parameter zusammengefasst, was die Menge stark verkleinern kann.

Der letzte Teil im Algorithmus ist der Layer Assignment Schritt. Dieser ordnet jedem Bildpunkt eine affine Bewegung zu. Der Layer Assignment Schritt benötigt zwei Dinge. Zum einen eine Funktion, die in der Lage ist, eine Zuordnung zu quantifizieren. Zum anderen ein Verfahren um aus der großen Menge von Lösungen – immerhin gibt es $4,7 \cdot 10^{44}$ mögliche Zuordnungen, bei einer Videosequenz von fünf Bildern, mit einer Auflösung von 320 mal 240 Bildpunkten und acht zuordenbaren affinen Parametern –, die Beste auszuwählen. Der Layer Assignment Schritt benutzt ein Näherungsverfahren, mit dem eine sehr gute Lösung, wenn auch nicht immer die Beste, ausgewählt wird. Abbildung 3.2 zeigt das Ergebnis des Layer Assignment Schritts.

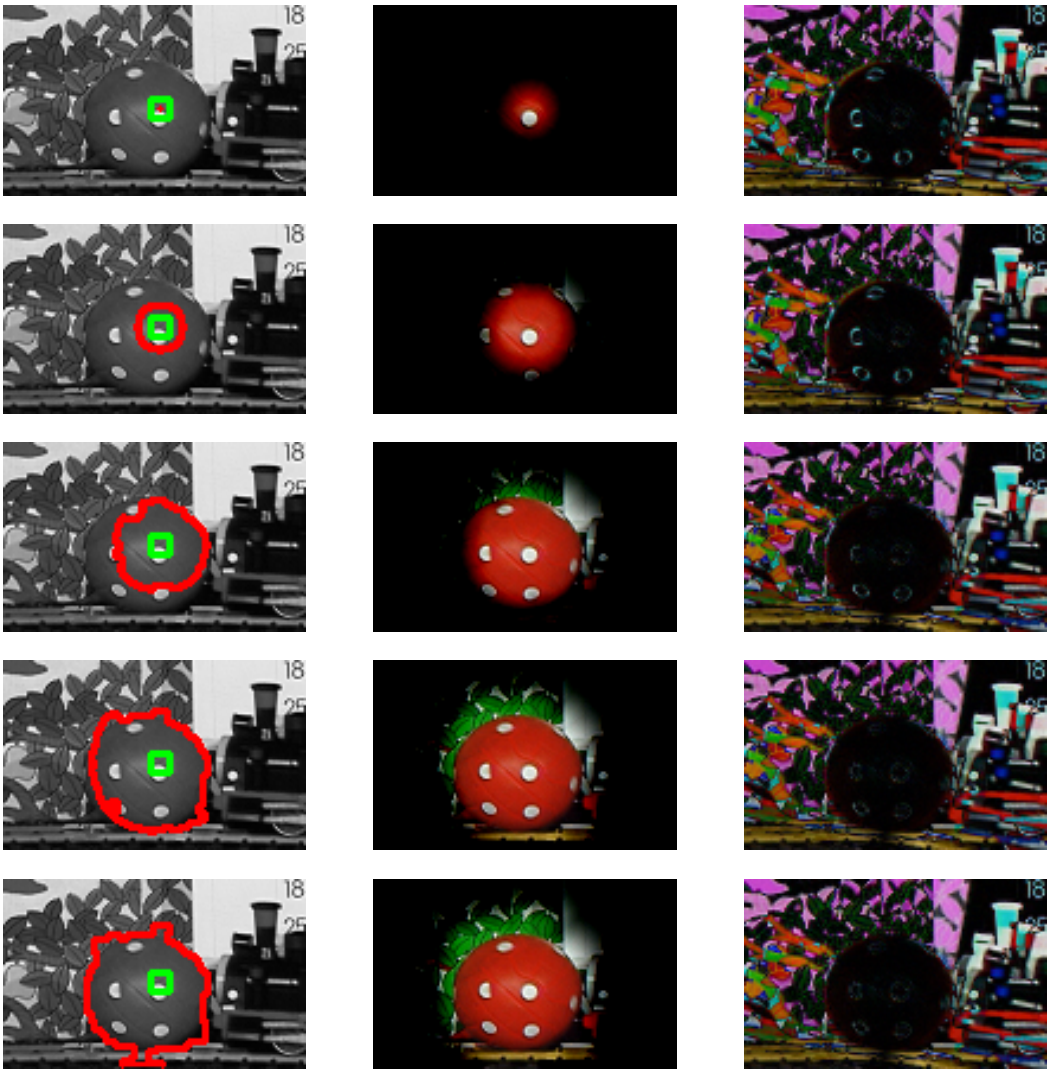


Abbildung 3.3: Ablauf des Region Growing Algorithmus am Beispiel des Ball Layers. Jede Zeile stellt einen Wachstumsschritt dar. Die linke Spalte zeigt mit der roten Eingrenzung die Region und mit dem grünen Quadrat den Start-Saatpunkt. Die mittlere Spalte veranschaulicht die Gewichtung der Nachbarbildpunkte. Je schwärzer ein Bildpunkt wird, umso geringer ist die Chance, in die Region aufgenommen zu werden. Die rechte Spalte zeigt die absolute Differenz zwischen dem Referenzbild und dem affin transformierten Bild. In der rechten Spalte erkennt man, wie die Differenzen im Ball Layer immer kleiner werden – was auf die verfeinerten affinen Parameter zurückzuführen ist.

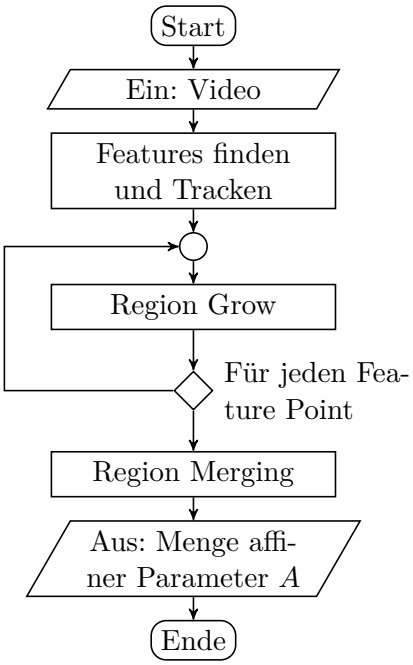



Abbildung 3.4: Ablaufdiagramm des Feature Extraction Schrittes.

## 3.2 Layer Extraction Schritt

Das Ziel dieses Schritts ist eine vollständige, heterogene Menge affiner Parameter, wie sie im Video vorkommen. Abbildung 3.4 stellt den Ablauf schematisch dar. Im Folgenden wird auf die einzelnen Teilschritte dieses Ablaufes eingegangen.

Ein Fenster, in dessen Zentrum ein Feature ist, wird *Featurefenster* genannt. Die affinen Parameter kann man näherungsweise suchen, indem die beste Übereinstimmung eines Featurefensters im nächsten Bild, durch Variieren der affinen Parameter, gefunden wird. Die Parameter sind zunächst ungenau, aber sie können schrittweise durch *Region Growing* verfeinert werden. Dabei werden in jeder Region Growing Wachstumsphase die umliegenden Bildpunkte überprüft, ob sie die Region unterstützen. Durch eine immer größer werdende Region können die affinen Parameter immer genauer bestimmt werden.

### 3.2.1 Region Growing

Die Eingabe für den Algorithmus ist ein Saatpunkt (Featurepoint), mit den dazugehörigen affinen Bewegungsparametern, vom KLT Tracker. Die Ausgabe des Algorithmus ist eine gewachsene Region und dazu die verfeinerten affinen Bewegungsparameter.

Der Algorithmus ist iterativ. In der Wachstumsphase werden die Nachbarbildpunkte der Region überprüft, ob sie die Region unterstützen. Wenn ja, dann werden sie mit ihr verschmolzen. Ein Bildpunkt unterstützt eine Region, wenn die affinen Parameter der

**Input** : Featurepoint $p$ mit dazugehörigen anfangs affinen Parametern $a$
**Output** : Genau bestimmte affine Parameter $a$ dieses Featurepoint

```
1  r ← p;                          /* Initialisiere Region mit Featurepoint */
2  gewachsen ← true;
3  while gewachsen do
4  |  gewichte ← Distanztransformation von r;
5  |  graph ← Erstelle Flussnetzwerk aus gewichte und a;
6  |  rneu ← Graph Schnitt von graph;
7  |  if größe von rneu > größe von r then
8  |  |  r ← rneu;
9  |  |  a ← Berechne affine Parameter der Region r;
10 |  else
11 |  |  gewachsen ← false;
12 |  end
13 end
```

Abbildung 3.5: Region Growing Algorithmus

Region besser zu ihm passen als ein experimentell ermittelter Threshold. Für die vergrößerte Region werden die affinen Parameter neu berechnet. Danach werden wieder die Nachbarbildpunkte der Region überprüft, ob sie die Region unterstützen. Unterstützen keine neuen Bildpunkte die Region, wird der Algorithmus beendet. Das Ergebnis ist eine Region mit genau bestimmten affinen Parametern. Abbildung 3.5 zeigt den Ablauf des Region Growings. Jeder Schritt des Algorithmus wird im Folgenden genauer erklärt.

### Distanztransformation

Welche Bildpunkte Nachbarbildpunkte der Region sind, entscheidet ein Distanzmaß. In dieser Arbeit wurde die Euklidische Distanz gewählt ($L2$ Distanz). Ein Parameter[1] entscheidet, bis zu welcher Distanz von der Region ein Bildpunkt als Nachbar in Frage kommt. Die Distanz eines Bildpunkts $p$ zur Region ist die kleinste Distanz von $p$ zu allen Bildpunkten der Region. Zur Berechnung der Distanzen wird ein Näherungsverfahren verwendet, wie es in [5] beschrieben wird und in der OpenCV Bibliothek [14] umgesetzt wurde. Dieser Algorithmus hat eine konstante Laufzeit $O(n)$, wobei $n$ die Anzahl der Bildpunkte ist. Es ist ein Näherungsverfahren, weil die Distanzen nicht exakt berechnet werden. Eine Distanz wird bei diesem Algorithmus immer als eine Kombination von Sprüngen berechnet. Es gibt horizontale, vertikale, diagonale und aus dem Schach bekannte Reitersprünge. Jeder Sprung verursacht Kosten. Die Distanz zu einem Bildpunkt ist die Summe der Kosten der Sprünge, die benötigt werden, um den Bildpunkt auf kürzestem Weg zu erreichen.

[1] Alle Parameter werden im Kapitel 4.1 erklärt.

### Das Flussnetzwerk für den minimalen Schnitt

Ob ein Bildpunkt die Region unterstützt oder nicht unterstützt, ist ein Zuordnungsproblem mit zwei Label. Dieses Zuordnungsproblem lässt sich als Energiefunktion, die in Gleichung (3.1) dargestellt ist, formulieren.

$$E(f) = E_{smooth}(f) + E_{data}(f) \tag{3.1}$$

$$E_{data}(f) = \sum_{p \in P} D_p(f_p) \tag{3.2}$$

$$E_{smooth}(f) = \sum_{\{p,q\} \in N} V_{\{p,q\}}(f_p, f_q) \tag{3.3}$$

Die Datenkosten $D_p$ für einen Bildpunkt $p$ sind in den Gleichungen (3.4) und (3.5) dargestellt.

$$D_p(1) = |J(A \cdot p + d) - I(p)| \tag{3.4}$$

$$D_p(0) = w(p) \cdot \gamma \tag{3.5}$$

Dabei sind $D_p(1)$ die Datenkosten, wenn der Bildpunkt zur Region gehören soll, $D_p(0)$ die Datenkosten, wenn er nicht zur Region gehören soll. $I(p)$ ist die Intensität des Bildpunkts $p$ im Bild $I$. $J(A \cdot p + d)$ ist die Intensität des Bildpunkts $p$ im Bild $J$, wobei der Bildpunkt um die affinen Parameter der Region transformiert wurde. $w(p)$ ist eine Gewichtung, die durch die Gaußfunktion realisiert wurde. Dabei werden Bildpunkte innerhalb der Region mit dem Faktor 1.0 gewichtet. Nachbarbildpunkte der Region werden entsprechend ihrem Abstand zur Region gewichtet. Das heißt, je größer die Distanz zur Region wird, desto kleiner wird $D_p(0)$ und desto größer die Wahrscheinlichkeit, dass $p$ nicht zur Region gehört.

Die Smoothnesskosten $V(p, q)$ sind in Gleichung (3.6) dargestellt.

$$V(p,q) = \begin{cases} 4\lambda & \text{wenn } \max(|I(p) - I(q)|, |J(A \cdot p) - J(A \cdot q)|) < 4 \\ 2\lambda & \text{wenn } 4 \geq \max(|I(p) - I(q)|, |J(A \cdot p) - J(A \cdot q)|) < 8 \\ \lambda & \text{ansonsten} \end{cases} \tag{3.6}$$

Wenn die Intensitätsunterschiede zwischen zwei benachbarten Bildpunkten $p$ und $q$ relativ groß sind, dann verursacht es geringe Kosten, den einen Bildpunkt der Region zuzuordnen und den anderen nicht. Wenn hingegen die Intensitätsunterschiede relativ klein sind, dann verursacht es dagegen große Kosten. Die Intensitätsunterschiede werden im ersten Bild und in dem Bild, für das die affinen Parameter berechnet werden, bestimmt.

Ziel ist, die Energiefunktion (3.1) zu minimieren und damit eine Zuordnung $f$ der Label zu den Bildpunkten zu finden, welche die geringsten Kosten verursacht. Da in diesem Fall nur zwei Label in Frage kommen, lässt sich dieses Problem exakt und effizient durch das Verfahren des minimalen Schnitts lösen. Auf das Verfahren selbst wurde im Kapitel 2.4 eingegangen. Dort wird auch ein Näherungsverfahren vorgestellt, mit dem man ein Zuordnungsproblem mit mehr als zwei Label lösen kann. Im Folgenden wird beschrieben,

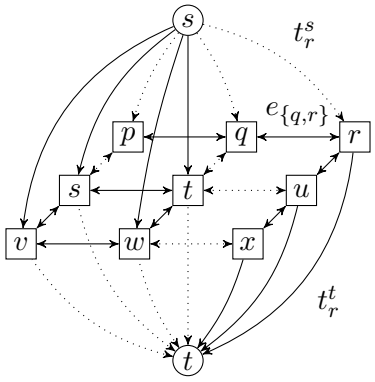



Abbildung 3.6: Ein Schnitt durch ein Flussnetzwerk, der neun Bildpunkte darauf testet, ob sie zur Region gehören. $\{p, q, r, u, x\}$ werden der Region zugeordnet und $\{s, t, v, w\}$ nicht. Die $t$ Kanten entsprechen Datenkosten, $t_r^s = D_r(1)$, $t_r^t = D_r(0)$, die $e$ Kanten den Smoothnesskosten, $e_{\{q,r\}} = V(q, r)$. Die punktierten Kanten sind vom Schnitt durchtrennt. Die Kapazität des Schnitts ist die Summe der Gewichte der punktierten Kanten.

wie das Flussnetzwerk erstellt wird, mit dessen Hilfe die Energiefunktion in Gleichung (3.1) exakt minimiert wird.

Für jeden Bildpunkt, der zur Region $R$ oder zu den Nachbarbildpunkten der Region gehört, wird ein Knoten in das Flussnetzwerk eingefügt. Diese Bildpunkte werden überprüft, ob sie die affinen Parameter der Region unterstützen oder nicht. Jeder Knoten bekommt eine Kante zur Quelle und eine zur Senke. Sind zwei Bildpunkte benachbart, dann wird zwischen den entsprechenden Knoten im Flussnetzwerk auch eine Kante eingefügt. Die Kanten von den Knoten zur Quelle entsprechen den Datenkosten, die entstehen, wenn ein Bildpunkt die Region unterstützt. Die Kanten von den Knoten zur Senke entsprechen den Datenkosten, wenn ein Bildpunkt die Region nicht unterstützt. Die Kanten zwischen den Knoten entsprechen den Smoothnesskosten, also jenen Kosten, die entstehen, wenn benachbarte Bildpunkte unterschiedliche Label erhalten. Abbildung 3.6 zeigt ein Beispiel eines Flussnetzwerks, wie es für neun Bildpunkte aussehen könnte. Ein minimaler Schnitt in diesem Flussnetzwerk löst das Zuordnungsproblem, indem er zwei Knotenmengen bildet. Bildpunkte, die zur Knotenmenge der Quelle gehören, unterstützen die Region nicht. Bildpunkte, die zur Knotenmenge der Senke gehören, unterstützen die Region. Der minimale Schnitt hat das Minimum der Energiefunktion (3.1) gefunden und das Zuordnungsproblem gelöst.

### 3.2.2 Region Merging

Ziel des Algorithmus ist nicht, die Regionen, deren Form oder Größe zu finden. Wie im einleitenden Kapitel 3.2 angeführt, soll das Ergebnis des Algorithmus eine vollständige und heterogene Menge von affinen Bewegungen sein, wie sie in der Eingabevideosequenz enthalten sind. Da meistens mehrere Features des KLT Trackers – und damit auch meh-

rere Regionen aus dem Region Growing Schritt – dasselbe Objekt in der Wirklichkeit charakterisieren, kommt eine affine Bewegung mehrmals vor. Ziel des Region Merging ist es, aus einer ursprünglich großen Menge affiner Bewegungsparameter vom Region Growing Schritt, in der noch viele affine Bewegungen fast identisch sind, eine kleine heterogene Menge zu erhalten. Diese Menge ist die Label Menge für den späteren Assignment Schritt.

Man kann nicht davon ausgehen, dass zwei affine Bewegungen exakt gleich sind. An irgendeiner Kommastelle unterscheiden sich die Parameter. Darum ist ein Maß für die Ähnlichkeit von zwei affinen Bewegungen erforderlich. Die Euklidische Distanz kann nicht verwendet werden, weil die Parameter des Translationsvektors eine andere Skalierung besitzen als die Parameter der Transformationsmatrix. Ob zwei affine Bewegungen vom Region Growing Schritt mit den dazugehörigen Regionen $R1$ und $R2$ identisch sind, wird folgendermaßen entschieden. Unterstützen – ob ein Bildpunkt die affinen Parameter unterstützt, wird in 3.2.1 erläutert – mehr als $99,5\%$ der Bildpunkte der Region $R1$ die affinen Parameter der Region $R2$ oder umgekehrt, dann werden die affinen Parameter von $R1$ und $R2$ als gleich angesehen und vereinigt. Vereinigt werden sie, indem von der Vereinigungsmenge der Bildpunkte der Regionen $R1 \cup R2$ die affinen Parameter neu berechnet werden – siehe dazu 2.2.1. Der Ablauf des Region Merging ist in Abbildung 3.7 dargestellt.

**Input** : Menge affiner Parameter und dazugehöriger Regionen
**Output** : Menge affiner Parameter und dazugehöriger Regionen

```
1  repeat
2  |  fertig ← true;
3  |  foreach Regionenpaar {R1, R2} do
4  |  |  if 99,5% der Bildpunkte von R1 unterstützen affine Parameter von R2 oder
   |  |  umgekehrt then
5  |  |  |  R3 ← R1 ∪ R2;
6  |  |  |  Berechnung der affinen Parameter von R3;
7  |  |  |  Löschen von R1 und R2 mit dazugehörigen affinen Parametern;
8  |  |  |  Einfügen von R3 mit dazugehörigen affinen Parametern;
9  |  |  |  fertig ← false;
10 |  |  |  break;                          /* Unterbricht foreach Schleife */
11 |  |  end
12 |  end
13 until fertig ;
```

Abbildung 3.7: Region Merging Algorithmus

## 3.3 Layer Assignment Schritt

Beim Layer Assignmen Schritt handelt es sich um ein klassisches Zuordnungsproblem, wie es in Computer Vision oft vorkommt. In dieser Arbeit entspricht ein *Label* einer affinen Abbildung. Das Ziel dieses Teilschritts ist, jedem Bildpunkt einer Videosequenz eine affine Bewegung zuzuordnen.

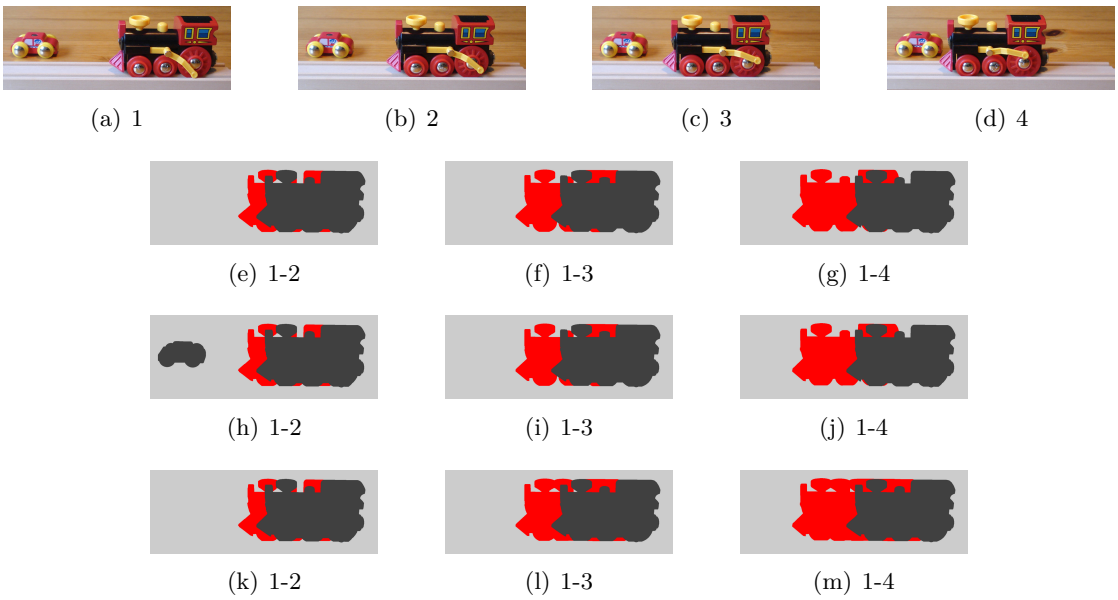
### 3.3.1 Idee

Eine affine Bewegung, wie sie im ersten Teil des Algorithmus gefunden wurde, beschreibt die Bewegung zwischen dem ersten und dem letzten Bild der Videosequenz. Die affine Bewegung zwischen aufeinanderfolgenden Bildern wird aus der affinen Bewegung zwischen dem ersten und dem letzten Bild interpoliert. Eine affine Bewegung zwischen dem ersten und zweiten Bild ist also eine lineare Interpolation der Bewegung zwischen dem ersten und letzten Bild. Das ist eine Vereinfachung, da eine Bewegung nicht stetig sein muss, sondern sich ändern kann. Wenn man zwischen wenigen aufeinanderfolgenden Bildern eine stetige Bewegung annimmt, heißt das, dass ein Bildpunkt, der eine affine Bewegung von Bild 1 und Bild 2 unterstützt, diese Bewegung auch zu Bild 3 unterstützen muss. Diese Annahme bietet den Vorteil, dass mehrere Bilder und damit mehr Informationen zum Zuordnen einer Bewegung zu einem Bildpunkt zur Verfügung stehen.

### 3.3.2 Energiefunktion

Jiangjan Xiao und Mubarak Shah haben eine Energiefunktion definiert, in der sie die Bewegung von mehreren Bildpaaren miteinander verzahnen, um damit das Zuordnungsergebnis zu verbessern. In Experimenten kann man diese Verbesserung auch praktisch feststellen. Bei der Verzahnung der Bildpaare wird die Zuordnung des Bildpaars $\{1, 2\}$ mit der Zuordnung des Bildpaars $\{1, 3\}$, die Zuordnung des Bildpaars $\{1, 3\}$ mit der von $\{1, 4\}$ und so weiter, in Beziehung gesetzt. Einem Bildpunkt, dem in einem Bildpaar ein bestimmter Label zugeordnet wird, darf in einem anderen Bildpaar nur derselbe Label oder der Verdeckungslabel zugeordnet werden. Abbildung 3.8 veranschaulicht anhand eines einfachen Beispiels von gültigen und ungültigen Zuordnungen diese Idee der Verzahnung der Bildpaare. Die Verzahnung der Bildpaare $\{1, 2\}$, $\{1, 3\}$ und $\{1, 4\}$, welche das Ergebnis in (e), (f) und (g) liefert, ist gültig. Die Verzahnung, welche das Ergebnis in (h), (i) und (j) liefert, ist ungültig und wird von der Energiefunktion gesperrt. Denn in der Zuordnung (h) hat der Bereich mit dem Spielzeugauto einen anderen Label, als in den Zuordnungen (i) und (j), was laut Definition der Energiefunktion nicht sein darf.

Jiangjan Xiao und Mubarak Shah haben den Verdeckungen, die in einem Video auftreten können, besondere Aufmerksamkeit geschenkt. Damit tragen sie der Tatsache Rechnung, dass nicht jedem Bildpunkt in einem Video eine Bewegung zugeordnet werden kann. Wenn ein Bildpunkt zwischen aufeinanderfolgenden Bildern in einem Bild nicht sichtbar ist, da er von einem Objekt verdeckt wird, kann diesem Bildpunkt keine Bewegung zugeordnet werden. In ihren beiden Arbeiten [35, 36] behandeln sie Verdeckungen unterschiedlich. In der Arbeit [35] erstellen sie zwei Bedingungen für Verdeckungen.



(a) 1 (b) 2 (c) 3 (d) 4

(e) 1-2 (f) 1-3 (g) 1-4

(h) 1-2 (i) 1-3 (j) 1-4

(k) 1-2 (l) 1-3 (m) 1-4

Abbildung 3.8: Die Bilder a, b, c und d zeigen einen Spielzeugzug, der sich von rechts nach links bewegt. Die Bilder über den Bildunterschriften e, f und g zeigen Bewegungszuordnungen zwischen den Bildpaaren $\{1, 2\}$, $\{1, 3\}$ und $\{1, 4\}$. Die Zuordnung wird in jedem Bildpaar auf das Referenzbild – das Bild mit der Nummer 1 – bezogen. Die roten Bereiche in den Zuordnungsbildern stellen Bildpunkte dar, die nicht zugeordnet werden konnten, weil sie im zweiten Bild des Bildpaares nicht sichtbar sind. Die Zuordnungen sind zum besseren Verständnis vereinfacht dargestellt. Die Rotationsbewegung der Räder und die Bewegung der Kolbenstange werden vernachlässigt.

1. Wenn ein Bildpunkt im Bildpaar $\{1, j\}$ den Verdeckungslabel $f_\zeta$ erhält, dann behält er diesen auch im Bildpaar $\{1, j+1\}$ bei.

2. Wenn ein Bildpunkt im Bildpaar $\{1, j\}$ den Label $f_p$ erhält, dann darf er im Bildpaar $\{1, j+1\}$ nur den Label $f_p$ oder den Verdeckungslabel $f_\zeta$ erhalten.

Ein Beispiel für Zuordnungen, welche diese beiden Bedingungen erfüllen, ist in Abbildung 3.8 in (k), (l) und (m) gegeben. In der darauf folgenden Arbeit [36] werden Verdeckungen so behandelt, wie es der Wirklichkeit eher entspricht. Wenn sich ein Objekt in eine Richtung bewegt, dann verdeckt es nicht nur immer weitere Bildpunkte, es erscheinen auch wieder Bildpunkte, die zuvor verdeckt waren. Abbildung 3.8 zeigt ein Beispiel in den Zuordnungen (e), (f) und (g). Man erkennt den Fortschritt gegenüber den Verdeckungen in den Zuordnungen (k), (l) und (m). Dazu haben sie die erste Bedingung gestrichen und die zweite Bedingung geändert zu: Wenn ein Bildpunkt im Bildpaar $\{i, j\}$ den Label $f_p \neq f_\zeta$ erhält, dann darf er im Bildpaar $\{i, k\}$ den Label $f_p$ behalten oder den Verdeckungslabel $f_\zeta$ erhalten. Bei dieser Bedingung gilt: $i \neq j \neq k$. Ein Bildpunkt kann also verdeckt werden und nach der Verdeckung wieder die ursprüngliche Bewegung – also den ursprünglichen Label – annehmen.

Um die Verzahnung der Bildpaare und die Verdeckungen im Zuordnungsproblem abzubilden, definieren sie eine Energiefunktion mit vier Termen: einem Datenterm, einem Smoothnessterm, einem Verdeckungsterm und einem generellen Verdeckungsterm. Die Energiefunktion ist in Gleichung (3.7) dargestellt.

$$E = \sum_{j=1}^{n-1} \Big( E_{sm_j}(f) + E_{d_j}(f) + E_{oc_j}(f) \Big) + \sum_{j=1}^{n-2} E_{g_j}(f) \tag{3.7}$$

Dabei ist $j$ ein Index für das Bild in der Videosequenz und $n$ die Anzahl der Bilder der Videosequenz. $f$ ist eine Zuordnungsfunktion, die jedem Bildpunkt eine affine Abbildung zuordnet. $E_{sm_j}$ sind die Smoothnesskosten für das Bildpaar $\{1, j+1\}$. Das erste Bild der Videosequenz ist das Referenzbild. Die Smoothnesskosten sind in Gleichung (3.8) dargestellt.

$$E_{sm_j}(f) = \sum_{i=0}^{1} \left( \sum_{\{p_j, q_j\} \in N} \Big( V\left(p_{j,i}, q_{j,i}\right) \cdot T(f_{p_j} \neq f_{q_j}) \Big) \right) \tag{3.8}$$

Dabei ist $N$ die Menge der benachbarten Bildpunkte. $T$ ist 1, wenn sein Argument wahr ist und 0 anderenfalls. $E_{d_j}$ sind die Datenkosten und in Gleichung (3.9) dargestellt.

$$E_{d_j}(f) = \sum_{p_j \in P_j} \Big( D(p_j, f_{p_j}) \cdot T(f_{p_j} \neq f_\zeta) \Big) \tag{3.9}$$

Dabei ist $f_\zeta$ der Verdeckungslabel. $P_j$ sind die Bildpunkte des Bildes $j$. $E_{oc_j}$ sind die Verdeckungskosten[2] und in Gleichung (3.10) dargestellt.

$$E_{oc_j}(f) = \sum_{p_j \in P_j} \Big( D_\zeta \cdot T(f_{p_j} = f_\zeta) \Big) \tag{3.10}$$

[2] Aus dem Englischen: Occlusion = Verdeckung. Deshalb die Abkürzung $E_{oc}$

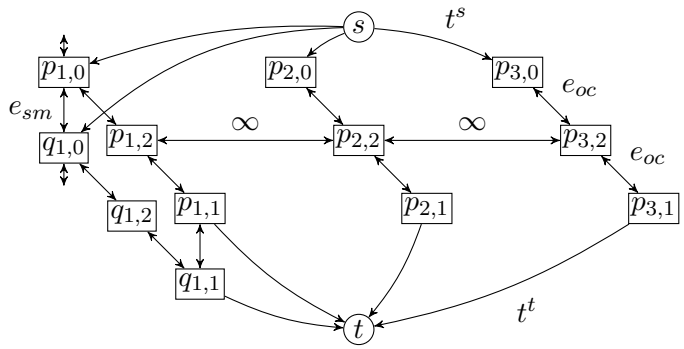



Abbildung 3.9: Das Flussnetzwerk, wie es in [36] verwendet wird, um die Energiefunktion (3.7) zu minimieren. Jeder Bildpunkt wird durch drei Knoten repräsentiert. $t$ - Kanten entsprechen den Datenkosten, $e_{sm}$ den Smoothnesskosten bei benachbarten Bildpunkten und $e_{oc}$ den Verdeckungskosten. Die erste Zahl im Index eines Bildpunkts entspricht der Variable $j$ in der Energiefunktion. Durch die $\infty$ Kanten ist es nicht möglich, dass zum Beispiel einem Bildpunkt $p$ im Bildpaar $\{1,2\}$ ein anderer Label als im Bildpaar $\{1,3\}$ zugeordnet werden kann. Er kann also nur den Label behalten oder den Verdeckungslabel annehmen.

Dabei ist $D_\zeta$ ein Parameter, der den Kosten entspricht, wenn dem Bildpunkt der Verdeckungslabel zugeordnet wird. $E_{g_j}$ sind die Verdeckungskosten zwischen Bildpaaren und in Gleichung (3.11) dargestellt.

$$E_{g_j}(f) = \sum_{\{p_j,p_k\}\in P} \Big(\infty \cdot T(f_{p_j} \neq f_\zeta \wedge f_{p_k} \neq f_\zeta \wedge f_{p_j} \neq f_{p_k} \wedge j \neq k)\Big) \tag{3.11}$$

Wenn einem Bildpunkt in einem Bildpaar ein Label zugeordnet wird, dann darf dieser Bildpunkt in einem anderen Bildpaar nur mehr diesen einen Label oder den Verdeckungslabel bekommen. Jede andere Zuordnung verursacht unendliche Kosten. Im Flussnetzwerk werden diese Kosten durch die $\infty$ - Kanten dargestellt. Das Flussnetzwerk, das diese Energiefunktionen abbilden soll, ist in Abbildung 3.9 dargestellt.

### 3.3.3 Optimierung

In der Arbeit von Jiangjan Xiao und Mubarak Shah [36] fehlen Angaben, wie sie die Datenkosten $D(p)$ und die Smoothnesskosten $V(p,q)$ bestimmen. Darum wurde versucht, die Kosten so zu berechnen, wie sie es in ihrer vorangegangenen Arbeit [35] getan haben. Die Smoothnesskosten $V(p,q)$ sind in Gleichung (3.12) dargestellt – man beachte, dass der Translationsanteil der affinen Bewegung in der Transformationsmatrix enthalten ist.

$$V(p,q) = \begin{cases} 4\lambda & \text{wenn } \max(|I_1(p) - I_1(q)|, |I_{j+1}(A_l \cdot p) - I_{j+1}(A_l \cdot q)|) < 4 \\ 2\lambda & \text{wenn } 4 \geq \max(|I_1(p) - I_1(q)|, |I_{j+1}(A_l \cdot p) - I_{j+1}(A_l \cdot q)|) < 8 \\ \lambda & \text{ansonsten} \end{cases} \tag{3.12}$$

Dabei ist $A_l$ entweder die affine Abbildung des Labels $\alpha$, wenn $i = 0$ ist, oder die affine Abbildung des Labels $f_p$, wenn $i = 1$ ist.

Diese Gleichung verursacht einige Schwierigkeiten bei der Berechnung von Gleichung (3.8). Erstens ist der Label $\alpha$ nur eine Variable im Expansion Move Algorithmus und deshalb nur diesem bekannt. Bei der Berechnung der Energie einer bestimmten Zuordnung gibt es die Variable $\alpha$ nicht. Zweitens hat der Verdeckungslabel $f_\zeta$ keine affine Bewegung. Bei einem verdeckten Bildpunkt $f_p = f_\zeta$ kann also $V(p, q)$ nicht bestimmt werden, weil es keine affine Bewegung $A_\zeta$ gibt. Drittens muss für den Expansion Move Algorithmus die Gleichung $V(p, q) = V(q, p)$ erfüllt sein. Diese Gleichung wird nicht erfüllt, da bei der Berechnung von $V(p, q)$ die affine Abbildung $A_{f_p}$ verwendet wird und bei der Berechnung von $V(q, p)$ die affine Abbildung $A_{f_q}$.

Aus diesen Gründen werden in dieser Arbeit die Smoothnesskosten berechnet, wie sie in Gleichung (3.13) dargestellt sind.

$$V(p,q) = \begin{cases} 4\lambda & \text{wenn } |I_1(p) - I_1(q)| < 4 \\ 2\lambda & \text{wenn } 4 \geq |I_1(p) - I_1(q)| < 8 \\ \lambda & \text{ansonsten} \end{cases} \tag{3.13}$$

Die Datenkosten werden von J. Xiao und M. Shah damit beschrieben, dass sie dafür den quadratischen Abstand eines Bildpunkts zwischen Referenzbild und dem transformierten Bild berechnen. In dieser Arbeit wird die absolute Differenz gewählt, weil diese etwas unempfindlicher auf Bildrauschen reagiert. Diese ist in Gleichung (3.14) dargestellt.

$$D(p, f_p) = |I_{j+1}(A_{f_p} \cdot p) - I_1(p)| \tag{3.14}$$

Dabei ist $I_j(p)$ die Intensität des Bildes $j$ an den Koordinaten von $p$. $A_f$ ist die Transformationsmatrix der affinen Abbildung $f$.

Wie in den Arbeiten [35, 36] angegeben, wird der Expansion Move Algorithmus verwendet, um die Energie zu minimieren. Beim Expansion Move Algorithmus ist die Energie der Zuordnung, die sich durch einen Expansion Move Schritt ergibt, gleich der Kapazität des minimalen Schnitts dieses Expansion Move Schritts [10]. Diese Gleichung $|C| = E(f^C)$ wird aus unterschiedlichen Gründen nicht erfüllt. Ein Grund ist, dass wenn zwei benachbarte Bildpunkte $\{p, q\} \in N$ einen unterschiedlichen Label $f_p \neq f_q$ haben, dann treten in der Energiefunktion (3.8) zweimal Smooth Kosten $V(p, q)$ für diese benachbarten Bildpunkte auf, nämlich für $i = 0$ und für $i = 1$. Angenommen der benachbarte Bildpunkt $q$ bekommt durch den minimalen Schnitt den Verdeckungslabel $f_\zeta$ zugeordnet, dann wird nur eine Smoothnesskante anstatt zwei durchtrennt. Nur eine Smoothnesskante für diese benachbarten Bildpunkte trägt zur Kapazität des Schnitts bei. Abbildung 3.10 zeigt das anschaulich. Aus diesem Grund wurde in dieser Arbeit die Smoothnessenergie angepasst und in der Form verwendet, wie sie in Gleichung (3.15) dargestellt ist.

$$\begin{aligned} E_{sm_j}(f) = & \sum_{\{p_j, q_j\} \in N} V\left(p_{j,i}, q_{j,i}\right) \cdot T(f_{p_j} \neq f_{q_j} \wedge f_{p_j} \neq f_\zeta) \\ & + \sum_{\{p_j, q_j\} \in N} V\left(p_{j,i}, q_{j,i}\right) \cdot T(f_{p_j} \neq f_{q_j} \wedge f_{q_j} \neq f_\zeta) \end{aligned} \tag{3.15}$$

Im Kapitel 2.5 wird erläutert, warum zwischen benachbarten Bildpunkten $\{p, q\}$ ein Hilfsknoten eingefügt wird, wenn $f_p \neq f_q$ ist. Dieser Hilfsknoten wird auf der Flussnetzwerkseite mit der Senke verwendet, also zwischen die Knoten $p_{j,1}$ und $q_{j,1}$ eingebaut. Abbildung 3.11 zeigt diese Konstruktion.

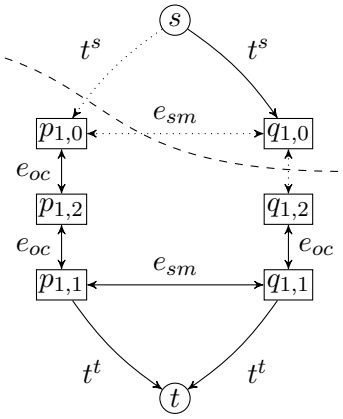
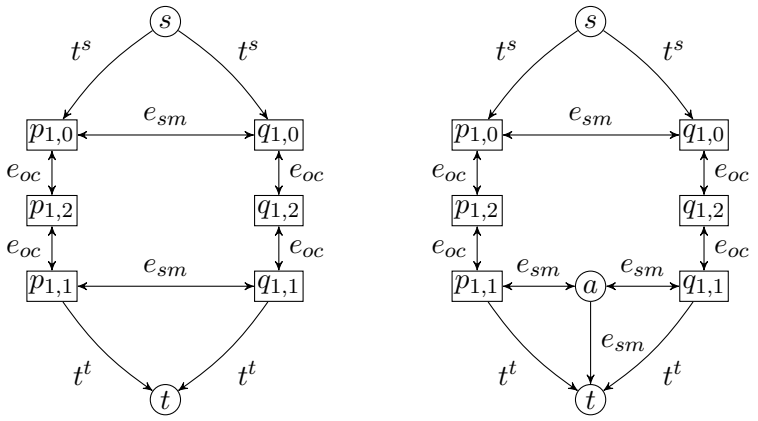



Abbildung 3.10: Ein kleiner Ausschnitt des Flussnetzwerkes aus Abbildung 3.9. Ein Schnitt, wie er durch die strichlierte Linie eingezeichnet ist, teilt dem Bildpunkt $p$ den Label $\alpha$ und dem Bildpunkt $q$ den Verdeckungslabel $l_\zeta$ zu. Die Kapazität des Schnitts ist die Summe der geteilten Kantengewichte $|C| = D(p, \alpha) + D_\zeta + V(p_{1,0}, q_{1,0})$. Die Energiefunktion (3.8) berücksichtigt aber zusätzlich noch die Kosten $V(p_{1,1}, q_{1,1})$.




(a) $p$ und $q$ haben denselben Label

(b) $p$ und $q$ haben unterschiedliche Label

Abbildung 3.11: Wenn benachbarte Bildpunkte unterschiedliche Label haben, dann ist, wie in (b), ein Hilfsknoten erforderlich. Damit werden alle drei Smoothnesskosten, die zwischen $p$ und $q$ entstehen können, berücksichtigt.

# 4 Implementierung

Die Arbeit wurde in der C++ Programmiersprache mit der freien Entwicklungsumgebung *Microsoft Visual C++ 2005 Express Edition* implementiert. Daneben wurde die Computer Vision Bibliothek von Intel *OpenCV* [14] verwendet, um die Bilder des Videos zu laden und die Ergebnisbilder anzuzeigen und abzuspeichern. Des Weiteren wird die Distanztransformationsfunktion – siehe Kapitel 3.2.1 – der OpenCV Bibliothek verwendet. Für den minimalen Schnitt Algorithmus wurden die Quelldateien von Yuri Boykov und Vladimir Kolmogorov verwendet. Ihr Algorithmus ist in der Arbeit [6] beschrieben. Zum Tracken der Featurepoints wird der KLT-Tracker von Takeo Kanade, Bruce D. Lucas und Carlo Tomasi verwendet. Ihr Algorithmus ist in den Arbeiten [27, 15] beschrieben. Ein Teil ihres Algorithmus beschäftigt sich mit dem Berechnen affiner Parameter – siehe Kapitel 2.2.1 – eines Featurefensters. Dieser Algorithmusteil wurde in dieser Arbeit für beliebige Bildpunktmengen angepasst.

Das Programm wurde auf einem Windows XP Betriebssystem entwickelt. Die verwendete Bibliothek OpenCV ist Open Source und auf vielen Plattformen verwendbar. Der KLT Tracker und der minimale Schnitt Algorithmus liegen im Quellcode vor. Aus diesen Gründen lässt sich diese Arbeit auch auf anderen Betriebssystemen kompilieren.

Das Programm benutzt nur einen Thread für den Algorithmus. Da die heute gängigen Prozessortypen üblicherweise mehrere Verarbeitungskerne besitzen, könnte man die Laufzeit verbessern, indem Teile des Algorithmus parallelisiert werden. Das Region Growing der Featurepoints kann parallelisiert werden, weil das Region Growing eines Featurepoint unabhängig ist vom Region Growing eines anderen Featurepoint. Auch der Region Merging Schritt bietet Möglichkeiten zur Parallelisierung.

## 4.1 Eingabe und Ausgabe des Programms

Das Programm *Lay.exe* wird in der Kommandozeile gestartet und erwartet als Aufrufparameter eine Initialisierungsdatei. Ein Beispiel dieser Initialisierungsdatei ist in Abbildung 4.1 dargestellt. Der Parameter *videopath* bestimmt den Dateinamen inklusive Pfad zu den Eingabebildern. Das Programm erweitert diesen Pfad um den Index und den Datentypen *ppm*. Als erstes Eingabebild erwartet das Programm für das Beispiel aus Abbildung 4.1 eine Datei mit dem Namen *mobil0.ppm*. Die Ausgabedateien werden in das Verzeichnis geschrieben, das durch den Parameter *outpath* angegeben wird.

Der Parameter *numberofimages* legt die Anzahl der Eingabebilder fest. Zwischen dem ersten und letzten Bild dieser Sequenz ermittelt das Programm die affinen Bewegungen. Zwischen den einzelnen Bildern werden die affinen Bewegungen linear interpoliert. Deshalb sollte dieser Parameter nicht zu groß gewählt werden, da sonst die Annahme einer linearen Bewegung nicht mehr erfüllt wird.

```
# Allgemeine Parameter
videopath            D:\MagArbeit\Datasets\mobile\mobil
outpath              D:\Temp\fertig_neu\mobil
numberofimages       5
startindex           0
displayprogress      0

# KLT Tracker
featureanz         100
windowsize_trans    11
windowsize          19
grad_sigma           1.0
smooth_sigma_fact    0.1

# Fuer den Feature Extraction Schritt
lambdaF             22.0
fwhm                 5.0
threshold           20.0
gaussborder          6.0

# Fuer den Layer Assignment Schritt
lambdaL             30.0
occlusion           15.0
```

Abbildung 4.1: Beispiel einer Initialisierungsdatei.

Der Parameter *startindex* bestimmt den Index des ersten Bildes der Sequenz, das geladen werden soll. Der Parameter *displayprogress* bestimmt, ob die einzelnen Wachstumsschritte des Region Growing Schritts abgespeichert werden sollen. Das verlangsamt die gesamte Laufzeit des Programms. Das Verhalten des KLT Trackers kann mit folgenden fünf Parametern eingestellt werden.

- featureanz: Die Anzahl der Featurepoints, die der KLT Tracker über die Bildersequenz zu *tracken* versucht. Die Features, die erfolgreich getrackt werden konnten, dienen später als Saatpunkte für den Region Growing Schritt.
- windowsize: Bestimmt die Breite und Höhe des Fensters, mit dem die affine Konsistenz geprüft wird. Das Fenster wird quadratisch festgelegt. Beim KLT Tracker heißen diese Parameter *affine_window_width* und *affine_window_height*.
- windowsize_trans: Legt die Größe des Fensters zum Tracken eines Features fest. Der KLT Tracker verwendet zum Tracken der Features ein Fenster mit festen Seitenlängen. Große Fenster erhöhen den Trackingerfolg, haben aber den Nachteil, dass Tiefenunstetigkeiten an Objektgrenzen innerhalb des Fensters häufiger auftreten. Dieser Parameter entspricht den Parametern *window_width* und *window_height* beim KLT Tracker.
- grad_sigma: Die Standardabweichung der Gaußfunktion in Pixel, zur Berechnung

der Bildgradienten.

- smooth_sigma_fact: Parameter zum Weichzeichnen der Bilder.

Der Region Growing Schritt benötigt die meisten Parameter.

- fwhm – Full Width Half Maximum: Bestimmt die Form der Gauß Glockenkurve. *fwhm* bestimmt die Breite der Glockenkurve bei der halben Höhe. Dieser Parameter wird im Region Growing Schritt benutzt, wo eine Gauß Gewichtung – siehe Gleichung (3.5) – der Bildpunkte vorgenommen wird. Bildpunkte in der Nähe der Region werden eher zur Region dazugefügt als Bildpunkte in größerer Entfernung.
- gaussborder: Definiert den Abstand von der Region, bis zu dem die Bildpunkte überprüft werden, ob sie die Region unterstützen oder nicht.
- threshold: Wert von $\gamma$ in der Gleichung (3.5).
- lambdaF: Wert von $\lambda$ in der Gleichung (3.6).

Der Layer Assignment Schritt, der die gefundenen affinen Bewegungen den Bildpunkten zuordnet, benötigt nur noch zwei Parameter.

- lambdaL: Wert von $\lambda$ in der Gleichung (3.13).
- occlusion: Wert von $D_\zeta$ in der Gleichung (3.10).

Ein Vorteil ist, wenn nur wenige Parameter vom Benutzer des Programms festzulegen sind. Je mehr Parameter, desto mehr Kombinationsmöglichkeiten ergeben sich und desto mehr Resultate unterschiedlicher Qualität können bei gleichem Videomaterial erzeugt werden. Daraus ergibt sich die Frage, ob es die optimalen Parameterwerte für alle möglichen Eingabevideos gibt. Und wenn es die optimalen Werte nicht gibt, lässt sich ein Verfahren angeben, das die optimalen Werte für ein bestimmtes Eingabevideo findet?

Die Ausgabe des Programms besteht zum einen aus Bildern, welche die Zuordnung der affinen Parameter zu den einzelnen Bildpunkten des Videos abbildet. Zum anderen werden die Zwischenergebnisse vom KLT Tracker und vom Layer Extraction Schritt abgespeichert. Als Text werden die Ergebnisse und Zwischenergebnisse in der Kommandozeile ausgegeben. Beispiele für Ausgaben sind im Kapitel 5 angeführt.

## 4.2 Beschreibung des Programms

Das Klassendiagramm in Abbildung 4.2 zeigt die Klassen des Programms *Lay.exe*, welche den Algorithmus implementieren. Es werden zu jeder Klasse nur die öffentlichen Funktionen dargestellt. Aus Platzgründen werden keine Aufrufparameter und Rückgabetypen im Klassendiagramm angegeben. Die Schnittstelle des KLT Trackers – in der Implementierung von Stan Birchfield und Thorsten Thormaehlen [2] – ist in keiner Klasse, sondern im globalen Namensbereich definiert. Trotzdem wird zur Strukturierung im Klassendiagramm eine Klasse *KLT* dargestellt. Das Verfahren des minimalen Schnitts

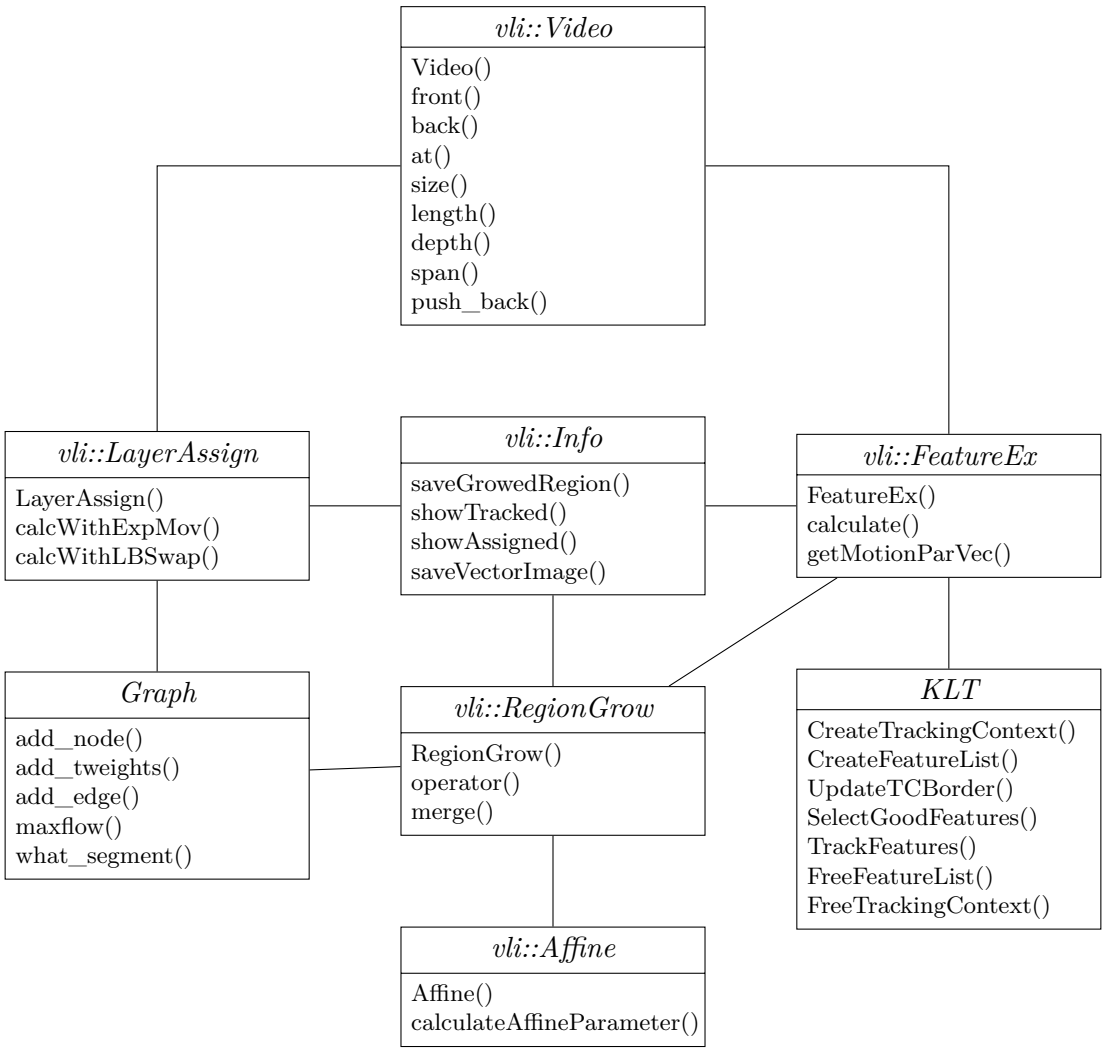



Abbildung 4.2: Das Klassendiagramm.

– in der Implementierung von Yuri Boykov und Vladimir Kolmogorov [7] – ist in der Klasse *Graph* abgebildet.

Alle Klassen, welche benutzerdefinierte Datentypen darstellen, werden wegen einer besseren Übersichtlichkeit nicht im Klassendiagramm dargestellt, sondern im Folgenden nur beschrieben. Es werden die benutzerdefinierten Datentypen mit den Namen *Parameter*, *Point*, *Region*, *MotionPar*, *MotionParVec*, *Layer*, *LayerVec*, *Zuordnung* und *ZuordnungArr* verwendet.

- Parameter: Dieser Datentyp enthält alle Parameter der Initialisierungsdatei.
- Point: Dieser Datentyp enthält zwei ganze Zahlen, die der $x$ und der $y$ Koordinate eines Punkts entsprechen.
- Region: Dieser Datentyp ist eine Menge von Punkten.
- MotionPar: Dieser Datentyp enthält eine affine Bewegung. Eine affine Bewegung besteht aus den acht Gleitkommazahlen $x$, $y$, $x2$, $y2$, $axx$, $axy$, $ayx$ und $ayy$. Der Translationsvektor ergibt durch $dx = x2 - x$ und $dy = y2 - y$.
- MotionParVec: Dieser Datentyp enthält eine Menge von affinen Bewegungen.
- Layer: Dieser Datentyp enthält eine Region und eine dazugehörige affine Bewegung.
- LayerVec: Dieser Datentyp enthält eine Menge Layer.
- Zuordnung: Dieser Datentyp enthält eine Menge von Punkten und zu jedem Punkt einen Index. Der Index bezieht sich auf eine affine Bewegung im Datentypen MotionParVec. Der Index ist eine ganze Zahl.
- ZuordnungArr: Dieser Datentyp enthält eine Menge von Zuordnungen.

### 4.2.1 Programmaufruf und Klassenbeschreibung

Der Einsprungspunkt des Programms ist die Funktion *main* in der Datei *main.cpp*. Dort wird in Folge die Initialisierungsdatei gelesen, die man als Aufrufargument angegeben hat. Die Initialisierungsdatei teilt dem Programm mit, an welchem Ort sich die Eingabebilder befinden. Die Eingabebilder werden in eine Instanz der Klasse *Video* geladen. Danach wird eine Instanz der Klasse *FeatureEx* mit einer Referenz zu diesem Video initialisiert. Durch die Funktion *calculate* werden die affinen Bewegungen berechnet und mit der Funktion *getMotionParVec* zurückgegeben. Eine Instanz der Klasse *LayerAssign* wird mit einer Referenz zum Video und den affinen Bewegungen initialisiert. Durch den Aufruf der Funktion *calcWithExpMov* werden die affinen Bewegungen jedem Bildpunkt zugeordnet. Die Zuordnung für ein Bildpaar wird im Datentyp *Zuordnung* gespeichert. Die Zuordnung für alle Bildpaare wird im Datentyp *ZuordnungArr* gespeichert. Die Ergebnisse werden über die Schnittstelle der Klasse *Info* ausgegeben.

Im Folgenden werden die einzelnen Klassen und ihre Funktionen genauer beschrieben. Alle Klassen, bis auf die fremden Klassen *Graph* und *KLT*, befinden sich im Namensbereich *Videolayer*. In diesem Namensbereich sind auch noch Hilfsfunktionen definiert, die keiner Klasse angehören. Diese werden später im Kapitel 4.2.2 beschrieben.

**FeatureEx** Die Aufgabe dieser Klasse ist, die affinen Bewegungen aus einem Video zu extrahieren. Sie implementiert den Algorithmusteil, der im Kapitel 3.2 beschrieben wurde. Eine Instanz dieser Klasse wird mit den Parametern aus der Initialisierungsdatei und mit dem Video initialisiert.
Durch den Aufruf der Funktion *calculate* werden die affinen Bewegungen berechnet. Die Funktion verwendet die Schnittstelle des KLT Trackers, um Saatpunkte zu bekommen. Danach wird mit Hilfe einer Instanz der Klasse *RegionGrow* jeder Saatpunkt gewachsen, um die affinen Bewegungen zu erhalten.
Mit dem Aufruf der Funktion *getMotionParVec* erhält man eine Variable vom Datentyp *MotionParVec*, der alle gefundenen affinen Bewegungen enthält.
Die Klasse verwendet die Funktionen der Klasse *Info*, um die Zwischenergebnisse abzuspeichern.

**KLT** In dieser Klasse sind die benutzten Funktionen des KLT Trackers dargestellt. Die Klasse FeatureEx benutzt diese Funktionen, um Saatpunkte für den Region Growing Schritt zu identifizieren. Der KLT Tracker ist im Kapitel 2.2 beschrieben.
Mit der Funktion *KLTCreateTrackingContext* wird eine Datenstruktur angelegt, in welcher die Parameter für den Tracker gespeichert sind. Mit der Funktion *KLTCreateFeatureList* wird eine Datenstruktur angelegt, in der die Featurepoints gespeichert werden. Die Funktion *KLTSelectGoodFeatures* sucht im ersten Bild nach Featurepoints, die das Trackingkriterium gut erfüllen – siehe Kapitel 2.2.2.
Die Funktion *KLTTrackFeatures* versucht die identifizierten Featurepoints im nächsten Bild wieder zu finden. Diese Funktion wird iterativ für jedes Bild aufgerufen, dabei wird die Featureliste jedes Mal aktualisiert. Geht ein Feature verloren, wird es nicht ersetzt, denn die Berechnung der affinen Parameter – im folgenden Region Growing Schritt – erfolgt zwischen dem ersten und letzten Bild der Sequenz.

**RegionGrow** Die Aufgabe dieser Klasse ist, einen Saatpunkt wachsen zu lassen und dabei die affinen Parameter in jedem Schritt neu zu berechnen. Die Klasse implementiert den Algorithmusteil, der im Kapitel 3.2.1 beschrieben wurde. Eine Instanz dieser Klasse wird mit dem ersten und letzten Bild der Videosequenz und den Parametern aus der Initialisierungsdatei initialisiert.
Die Funktion *operator()* erhält als Eingabe – im Datentyp *Layer* – einen Saatpunkt und die dazugehörige affine Bewegung. Die Ausgabe – auch im Datentyp *Layer* – der Funktion ist eine gewachsene Region mit den berechneten affinen Parametern. Der Saatpunkt und die dazugehörige affine Bewegung stammen vom KLT Tracker.
Außerdem implementiert die Klasse noch die Funktion *merge*, die von der Klasse *FeatureEx* verwendet wird, um zwei Regionen zu testen, ob sie zusammengelegt werden sollen.

Die Funktion *merge* wurde in dieser Klasse implementiert, weil sie dasselbe Flussnetzwerk wie der Region Growing Schritt verwendet.

**Affine** Die Aufgabe dieser Klasse ist, die affinen Parameter neu zu bestimmen. Eine Instanz dieser Klasse wird mit dem ersten und letzten Bild der Videosequenz initialisiert. Die Funktion *calculateAffineParameter* erhält die Region und die affine Bewegung aus der letzten Region Growing Iteration. Die Rückgabe der Funktion ist – im Datentyp *MotionPar* – eine neu bestimmte affine Bewegung. Die Berechnung der affinen Parameter ist im Kapitel 2.2.1 beschrieben. Weil die Berechnung umfangreich ist und viele Funktionen umfasst, wurde eine eigene Klasse entworfen. Bei den Funktionen handelt es sich um die Funktionen, die im KLT Tracker von Thorsten Thormaehlen implementiert wurden. Sie wurden in dieser Arbeit angepasst, damit die Berechnung nicht auf ein Fenster von fixer Seitenlänge zurückgreift, sondern eine Region beliebiger Größe benutzt.

**LayerAssign** Diese Klasse implementiert den Teil des Algorithmus, der im Kapitel 3.3 beschrieben wurde. Eine Instanz der Klasse wird mit dem Video, den Parametern der Initialisierungsdatei und den affinen Bewegungen – im Datentyp *MotionParVec* – initialisiert.
Durch den Aufruf der Funktion *calcWithExpMov* wird der Expansion Move Algorithmus gestartet und jedem Bildpunkt des Videos eine affine Bewegung zugeordnet. Die Funktion *calcWithLBSwap* startet den $\alpha\beta$ Swap Move Algorithmus, der dasselbe tut. Die Funktion *calcWithExpMov* ist im Gegensatz zur Funktion *calcWithLBSwap* korrekt in dem Sinne, dass die Kapazität des Schnitts der Energie der aktuellen Zuordnung entspricht.

**Graph** In dieser Klasse ist der Maxflow Algorithmus von Yuri Boykov und Vladimir Kolmogorov implementiert. Die Klasse *RegionGrow* verwendet diese Klasse zum Erstellen eines Flussnetzwerks, wie es im Kapitel 3.2.1 beschrieben wurde, um die Bildpunkte zu identifizieren, welche die Region unterstützen. Die Klasse *LayerAssign* verwendet diese Klasse zum Erstellen eines Flussnetzwerks, wie es im Kapitel 3.3.2 beschrieben wurde, um den Bildpunkten eine affine Bewegung, im Expansion Move Verfahren, zuzuordnen.
Mit der Funktion *add_node* wird ein Knoten in das Flussnetzwerk eingefügt. Mit dem Rückgabewert der Funktion kann man diesen Knoten später identifizieren.
Mit der Funktion *add_edge* wird eine Kante in das Flussnetzwerk eingetragen. Dieser Funktion übergibt man zwei Knoten und die Gewichte der Kanten in die eine und die andere Richtung.
Wenn das Flussnetzwerk fertig ist, wird mit der Funktion *maxflow* der minimale Schnitt berechnet. Der Rückgabewert der Funktion *maxflow* ist der Fluss des minimalen Schnitts.
Mit der Funktion *what_segment* kann man jeden Knoten testen, ob er zur Knotenmenge der Quelle oder der Senke gehört.

**Video** Eine Instanz dieser Klasse enthält alle Bilder eines Videos und bietet Zugriffsfunktionen an, die sich auf die Eigenschaften des Videos beziehen oder mit denen einzelne

Bilder aus dem Video geholt werden können. Die Bilder haben den Datentyp *IplImage*, das ist der Bild Datentyp der OpenCV Bibliothek.
Da an vielen Stellen im Programm das erste bzw. letzte Bild des Videos benötigt wird, wurden dafür die Funktionen *front* und *back* implementiert. Die Funktion *at* bietet einen willkürlichen Zugriff. Der Aufruf erfolgt über einen Index, der zwischen 1 und *length* liegen muss. Die Zugriffsfunktionen *front*, *back* und *at* liefern immer einen konstanten Zeiger auf das jeweilige Bild.
Die Funktion *size* liefert die Größe des Videos. Die Größe wird im Datentyp *CvSize* – der Datentyp ist in der OpenCV Bibliothek definiert – angegeben, der zwei ganze Zahlen *width* und *height* enthält.
Die Funktion *depth* liefert die Tiefe, in dem ein Bild des Videos vorliegt. Die Tiefe wird als ganze Zahl angegeben, die einer OpenCV Konstanten entspricht. Die Konstante *IPL_DEPTH_32F* bedeutet beispielsweise, dass die Bildpunkte als 32 Bit Fließkommazahl abgespeichert sind.
Die Funktion *span* liefert aus dem Video ein Teilvideo. Da vom Programm immer alle Bilder, die im Videopath vorhanden sind, geladen werden, wird mit dieser Funktion auf das Teilvideo zugegriffen, mit dem der Layer Extraction und der Layer Assignment Schritt initialisiert werden. Die Aufrufparameter der Funktion sind ein Startindex und eine Anzahl.
Mit der Funktion *push_back* wird ein Bild an das Ende des Videos angehängt. Diese Funktion wird zum Laden des Videos verwendet.

**Info** Mit den Funktionen dieser Klasse werden die Ergebnisse und Zwischenergebnisse auf der Festplatte abgespeichert. Die Ausgabe erfolgt in das Verzeichnis, das durch den Parameter *outpath* angegeben wurde.
Die Funktion *saveGrowedRegion* erwartet im Aufruf das erste und letzte Bild der Videosequenz und einen Layer. Das zweite Bild wird durch die affine Bewegung – die im Layer gespeichert ist – transformiert, um danach die absolute Differenz zum ersten Bild zu bilden. In dieses neue Bild wird die Region – auch im Layer enthalten – mit der Farbe Magenta sichtbar gemacht. Das Ergebnisbild wird abgespeichert. Diese Funktion wird von der Klasse *RegionGrow* verwendet, um die einzelnen Wachstumsschritte abzuspeichern. Dafür muss der Parameter *displayprogress* den Wert 1 haben. Die Funktion wird auch von der Klasse *FeatureEx* verwendet, um einerseits die Ergebnisse nach dem Region Growing Schritt und andererseits die Ergebnisse nach dem Region Merging Schritt abzuspeichern. Die Funktion *saveGrowedRegion* wird also in drei Arten verwendet. Darum teilt ein Aufrufparameter der Funktion mit, für welche Art sie aufgerufen wurde, damit sie entscheiden kann, wie sie die Ausgabedatei benennen muss.
Die Funktion *showTracked* wird von der Klasse *FeatureEx* verwendet und speichert das Ergebnis des KLT Trackers ab. Die Aufrufparameter sind das erste und letzte Bild der Sequenz und die Koordinaten der Featurepoints in diesen Bildern. Die Featurepoints sind im Datentyp *MotionParVec* gespeichert.
Die Funktion *showAssigned* wird von der Klasse *LayerAssign* verwendet und speichert das Ergebnis des Layer Assignment Schritts ab. Der Funktion wird die Zuordnung jedes

Bildes im Datentyp *ZuordnungArr* übergeben.
Die Funktion *saveVectorImage* speichert zwei Graustufenbilder, bei denen helle Bereiche viel Bewegung und dunkle Bereiche wenig Bewegung anzeigen. Das Bild *ZuordnungX_0.png* stellt die $x$ Komponente der Bewegungsvektoren dar, das Bild *ZuordnungY_0.png* die $y$ Komponente. Die 0 in diesem Beispiel deutet an, dass es sich um die Zuordnung des Bildpaares $\{1, 2\}$ handelt. Die Funktion wird von der Klasse *LayerAssign* aufgerufen, wenn der Layer Assignment Schritt fertig ist, um die Zuordnungen als Vektorbilder sichtbar zu machen. Die Funktion erwartet die Zuordnungen und die affinen Bewegungen. Die Zuordnungen werden im Datentyp *ZuordnungArr* übergeben, die affinen Bewegungen im Datentyp *MotionParVec*.

### 4.2.2 Hilfsfunktionen

Im Namensbereich Videolayer sind neben den oben beschriebenen Klassen auch noch einige Hilfsfunktionen definiert, die hier kurz beschrieben werden. Die Hilfsfunktionen dienen zum Beispiel zur Umwandlung von Bildern, zur Umwandlung eines Videos, zum Laden eines Videos oder zur Ausgabe von Zwischenergebnissen in die Kommandozeile. Eine Hilfsfunktion ist in keiner der oben beschriebenen Klassen definiert, weil sie semantisch in keine von ihnen passt, um die Klasse nicht mit zu vielen Funktionen zu überfüllen oder weil sie von mehreren Klassen benötigt wird, eine gemeinsame Basisklasse aber überdimensioniert wäre.

**computeAffineMappedPixel** Diese Funktion bekommt im Aufruf ein Bild, eine affine Bewegung und einen Punkt. Die Funktion transformiert den Punkt mit der affinen Bewegung und liefert den Farbwert des Eingabebildes an der transformierten Stelle.

**interpolate** Diese Funktion erhält im Aufruf ein Bild und eine Koordinate. Die Koordinate wird durch zwei Fließkommazahlen angegeben und stellt damit nicht exakt einen Bildpunkt dar. Diese Funktion ermittelt den Farbwert aus vier Bildpunkten in dem Verhältnis, in dem sie dem durch die Koordinate bestimmten Punkt nahe kommen. Die Funktion *computeAffineMappedPixel* verwendet diese Funktion.

**dataPenality** Diese Funktion wird von der Klasse RegionGrow und von der Klasse LayerAssign verwendet. Diese Funktion ermittelt die Datenkosten der Energiefunktion, die in den Gleichungen (3.4) und (3.14) spezifiziert sind.

**smoothPenality** Diese Funktion wird von der Klasse RegionGrow und von der Klasse LayerAssign verwendet. Diese Funktion ermittelt die Smoothnesskosten der Energiefunktion, die in den Gleichungen (3.6) und (3.13) spezifiziert sind.

**iplImageToUChar** Mit dieser Funktion wird ein Bild vom Datentyp *IplImage* in ein Datenformat umgewandelt, wie es die Funktionen des KLT Trackers benötigen.

**loadVideo** Diese Funktion erhält im Aufrufparameter einen Pfad und gibt im Ergebnis ein Video zurück.

**convertVideo** Diese Funktion konvertiert ein Video. Eingabeparameter sind ein Video und die gewünschte Tiefe des Ausgabevideos. Der Rückgabewert ist ein Video mit der gewünschten Tiefe.

**operator*** Der Multiplikationsoperator wurde spezialisiert, sodass eine affine Bewegung mit einer Gleitkommazahl multipliziert werden kann. Diese Funktion wird verwendet, um die affine Bewegung linear zu interpolieren.

**operator≪** Dieser Operator wurde spezialisiert, sodass eine Instanz der Klasse *FeatureEx*, eine Instanz des Datentyps *Layer* und *MotionPar*, als Text in der Konsole ausgegeben werden kann.

# 5 Ergebnisse

In diesem Kapitel wird auf die Ergebnisse verschiedener Testsequenzen eingegangen. Das Programm wird mit verschiedenen Parametern initialisiert und anhand der Ausgaben kann man erkennen, welchen Einfluss ein bestimmter Parameter hat. Bei den Testsequenzen – siehe Abbildung 5.1 – handelt es sich um Standardsequenzen, wie *Moving Car*, *Mobile & Calendar*, *Flower Garden* und *Tennis*. Die Sequenz *Selfrecorded Train* stammt vom Institut für Softwaretechnik und Interaktive Systeme. Die historischen Filmaufnahmen *Sequenz 2*, *Sequenz 3* und *Sequenz 6* aus den 1920er und 1930er Jahren stammen aus dem Projekt *Digital Formalism: The Vienna Vertov Collection.*

Beim Video *Moving Car* fährt das Auto in der Szene von links nach rechts. Sonst gibt es keine Objektbewegung. Die Kamera wird von Hand geführt, dadurch ruckeln die einzelnen Bilder etwas rauf und runter. Durch die Kamerabewegung bewegt sich die gesamte Szene etwas nach rechts.

Beim Video *Flower Garden* gibt es keine Objektbewegung. Die Kamera scheint eine lineare gleichförmige Bewegung von links nach rechts durchzuführen. Die Kamerarichtung bleibt unverändert. Man hat den Eindruck als ob man aus einem Zugfenster blickt.

Beim Video *Tennis* sind Kamera, Hintergrund und Tischplatte starr. Es bewegt sich der Ball und der Spieler. Im Videoausschnitt, der in dieser Arbeit untersucht wurde, bewegt sich der Ball in einer beschleunigten Bewegung nach unten und der Arm des Spielers nach oben.

Beim *Selfrecorded Train* gibt es nur eine Bewegung, nämlich die des Spielzeugzugs, der von rechts nach links fährt. Es gibt auch keine Kamerabewegung. Die Szene wird durch ein Weitwinkelobjektiv stark verzerrt. Darum ist die Bewegung des Zugs nicht geradlinig, sondern leicht gekrümmt.

Die drei historischen Russland-Videos sind Schwarzweiß Aufnahmen, deren Qualität durch ihr Alter etwas gelitten hat. Man sieht von Bild zu Bild unterschiedliche Flecken und Streifen. Manche Bilder einer Szene sind oft dunkler oder heller als die anderen Bilder der Szene. Im Video *Sequenz 2* sind zwei entgegenkommende Autos. Am rechten Szenenrand erscheint ein Auto, das, von der Kamera gesehen, wegfährt. Die Kamera ist starr. Vereinzelt sieht man Bewegungen der Menschen, vor allem des Mannes, der im Vordergrund zu sehen ist.

*Sequenz 3* zeigt Waggons eines Zugs, der sich in die Richtung der Kamera bewegt. Die Kamera ist starr, allerdings fällt es auf, dass die einzelnen Bilder etwas hüpfen, also nicht ganz in Übereinstimmung miteinander gebracht worden sind. Bei der *Sequenz 6* sieht man Eisenbahnwagons von rechts nach links über eine Brücke fahren.

Dieses Kapitel ist unterteilt und behandelt in jedem Unterkapitel die Parameter und Ausgaben der Zwischenschritte des Algorithmus. Im Folgenden wird auf die Ergebnisse des KLT Trackers, des Region Growing Schritts, des Region Merging Schritts und danach

Abbildung 5.1: Startbilder der verwendeten Testsequenzen. In der Reihenfolge von links oben nach rechts unten: *Moving Car*, *Mobile & Calendar*, *Flower Garden*, *Tennis*, *Selfrecorded Train*, *Sequenz 2*, *Sequenz 3* und *Sequenz 6*.

auf die Ergebnisse des Layer Assignment Schritts eingegangen. Begonnen wird mit den Ergebnissen des KLT Trackers.

## 5.1 KLT Tracker Ergebnis

Die meisten Parameter des KLT Trackers können ihren Defaultwert behalten. Die vier Parameter, welche man mit dem Programm *Lay* verändern kann, sind bereits im Kapitel 4.1 beschrieben worden. Damit *Lay* gute Ergebnisse erzielt, ist es zum einen erforderlich, dass jede Bewegung erfasst wird. Das heißt, es muss in jeder Bewegungsregion mindestens ein Featurepoint erkannt werden. Zweitens dürfen beim Tracken nur so viele Featurepoints verloren gehen, dass in jeder Bewegungsregion mindestens ein Featurepoint übrig bleibt. Mit dem Parameter *featureanz* kann man bestimmen, wie viele Featurepoints ausgewählt werden. Wenn eine Bewegungsregion keinen Featurepoint erhalten hat, kann man diesen Parameter so lange vergrößern, bis jede Bewegungsregion mindestens einen Featurepoint erhalten hat. Die schwierigere Aufgabe ist, diese Featurepoints erfolgreich über die Eingabesequenz zu tracken, da verloren gegangene Featurepoints nicht durch neue ersetzt werden dürfen. Der Grund dafür ist, dass im Region Growing Schritt eine affine Bewegung immer zwischen dem ersten und letzten Bild der Eingabesequenz berechnet wird.

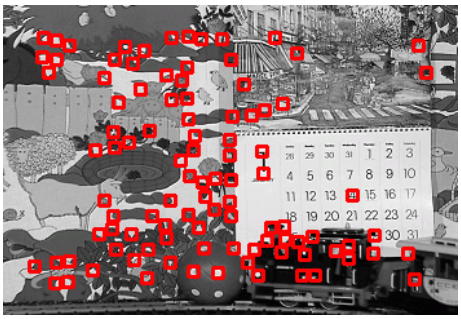
Das Problem bei der *Mobile & Calendar* Sequenz war, dass mit den Defaultwerten des KLT Trackers die Featurepoints im Kalender großteils verloren gingen. Damit diese Featurepoints erhalten werden, kann man die KLT Parameter anpassen. Als besonders nützlich haben sich dafür die Parameter *grad_sigma*, *smooth_sigmafact* und *windowsize_trans* erwiesen. In Abbildung 5.2 sieht man im Bildpaar (a) und (b) den Trackingerfolg mit den Defaultparametern. Um den Trackingerfolg im Kalenderbereich zu erhöhen, genügt eine Änderung des Parameters *smooth_sigma*, wie man im Bildpaar (c) und (d) sieht. Mit den Parametern *smooth_sigmafact* und *windowsize_trans* kann man ähnliche Ergebnisse erzielen.

(a) Bild 1

(b) Bild 5

(c) Bild 1

(d) Bild 5



(e) Bild 1

(f) Bild 5

Abbildung 5.2: Ergebnisse des KLT Trackers. Die linke Seite zeigt die gefundenen Features, die rechte Seite zeigt, welche davon erfolgreich über die gesamte Eingabesequenz getrackt werden konnten. Bei Bildpaar (a) und (b) hat der Parameter *grad_sigma* den Defaultwert von 1.0. Bei (c) und (d) den Wert 0.8 und bei (e) und (f) den Wert 1.2. Im Kalenderbereich sieht man die Auswirkungen.

## 5.2 Region Growing Ergebnis

Beim Region Growing werden alle Featurepoints vom KLT Tracker als Saatpunkte verwendet. Ein Saatpunkt ist die Startregion, die in jedem Iterationsschritt vergrößert wird, bis in einem Iterationsschritt keine neuen Bildpunkte mehr zur Region dazukommen. Während des Wachsens werden die affinen Parameter der Region immer neu berechnet, um so ziemlich exakte Werte für die affine Bewegung dieser Region zu erhalten. Hier gibt es ein paar Punkte, die berücksichtigt werden müssen, wenn man die Parameter einstellt.

Der Parameter *threshold* bestimmt, wie groß die Intensitätsunterschiede sein dürfen, sodass ein Bildpunkt noch zur Region aufgenommen wird. Dieser Parameter soll so gewählt werden, dass auch Bildpunkte aufgenommen werden, wo noch eine gewisse Intensitätsdifferenz vorhanden ist. Denn gerade in diesen Bildpunkten steckt die Information, die zur exakteren Bestimmung der affinen Parameter benötigt wird. Würde *threshold* so klein gewählt, dass nur Bildpunkte in die Region aufgenommen werden, die fast die gleiche Intensität in beiden Bildern haben, dann kommt bei der Neuberechnung der affinen Bewegung immer das gleiche Resultat heraus. Man erkennt dieses Verhalten daran, dass die Region nur in die homogenen Bereiche hineinwächst, während die texturierten Bereiche von der Region nicht berührt werden.

Es lässt sich nur schwer verhindern, dass eine Region in einen Bildbereich hineinwächst, der nicht mehr zu diesem Objekt dazugehört. Oft passiert das an einer nicht texturierten Stelle. Meistens ist das nicht schlimm, denn die Region ist dann oft schon groß genug, dass die affinen Parameter stabil sind. Bildpunkte aus anderen Bereichen können dann das Ergebnis nicht mehr beeinflussen. Abbildung 5.3 zeigt ein Beispiel einer stabilen Region, bei der die affinen Parameter nicht mehr verändert werden, auch wenn die Region in einen anderen Bereich hineinwächst. Wenn aber die Region klein ist, oder die Region Growing Parameter schlecht gewählt wurden, dann ist es möglich, dass das Ergebnis verfälscht wird, wenn die Region in einen anderen Bereich hineinwächst. Abbildung 5.4 zeigt ein Beispiel dafür. Es ist auch möglich, dass die ursprüngliche Region nichts mehr mit dem Ergebnis gemeinsam hat. Abbildung 5.5 zeigt ein Beispiel, bei dem affine Parameter bestimmt werden, die weder den Kalender, noch den Hintergrund richtig unterstützen. Das Ergebnis ist eher der Durchschnitt der affinen Parameter aus Hintergrund und Kalender. Und wenn einmal die Region eines Featurepoint innerhalb dieses Objekts nach außen wächst, dann wird es sehr wahrscheinlich für jeden anderen Featurepoint innerhalb dieses Objekts genau so sein. Das heißt, dass dieses Objekt nie durch die richtige affine Bewegung beschrieben wird. Und das führt dann auch zu keinem optimalen Ergebnis im Layer Assignment Schritt.

Die stabile Kalenderregion aus Abbildung 5.3 ist ein Ergebnis der Initialisierung, wie sie in der Abbildung 4.1 gezeigt wurde.

(a) Referenzbild

(b) Iterationsschritt 0

(c) Iterationsschritt 7

(d) Iterationsschritt 19

(e) Iterationsschritt 22

(f) Iterationsschritt 36

Abbildung 5.3: Region Growing eines Saatpunktes im Kalender. Die Region wächst über den Kalender hinaus. Ab dem siebten Iterationsschritt ist keine Veränderung der affinen Parameter mehr erkennbar. Dass die Region an beiden Seiten des Kalenders in den Hintergrund hinauswächst, ändert die affinen Parameter nicht mehr. Es werden nur mehr die homogenen Flächen außerhalb des Kalenders zur Region aufgenommen.

(a) Referenzbild

(b) Iterationsschritt 0

(c) Iterationsschritt 10

(d) Iterationsschritt 23

(e) Iterationsschritt 30

(f) Iterationsschritt 31

Abbildung 5.4: Region Growing eines Saatpunktes im Kalender. Die Region wächst über den Kalender hinaus. Deshalb werden die affinen Parameter schlecht bestimmt. Ob die affinen Parameter gut passen, erkennt man am Einfachsten, indem man einen Kalendertag von Bild zu Bild vergleicht. Man erkennt, dass die affinen Parameter im Ergebnis (f) schlechter bestimmt wurden als die Initialisierungsparameter des Saatpunktes in (b). Am besten stimmen die Parameter zwischen (c) und (d), hier ist von Schritt zu Schritt kaum eine Veränderung der affinen Parameter festzustellen.

(a) Iterationsschritt 7

(b) Iterationsschritt 33

(c) Iterationsschritt 46

(d) Iterationsschritt 61

(e) Iterationsschritt 78

(f) Iterationsschritt 119

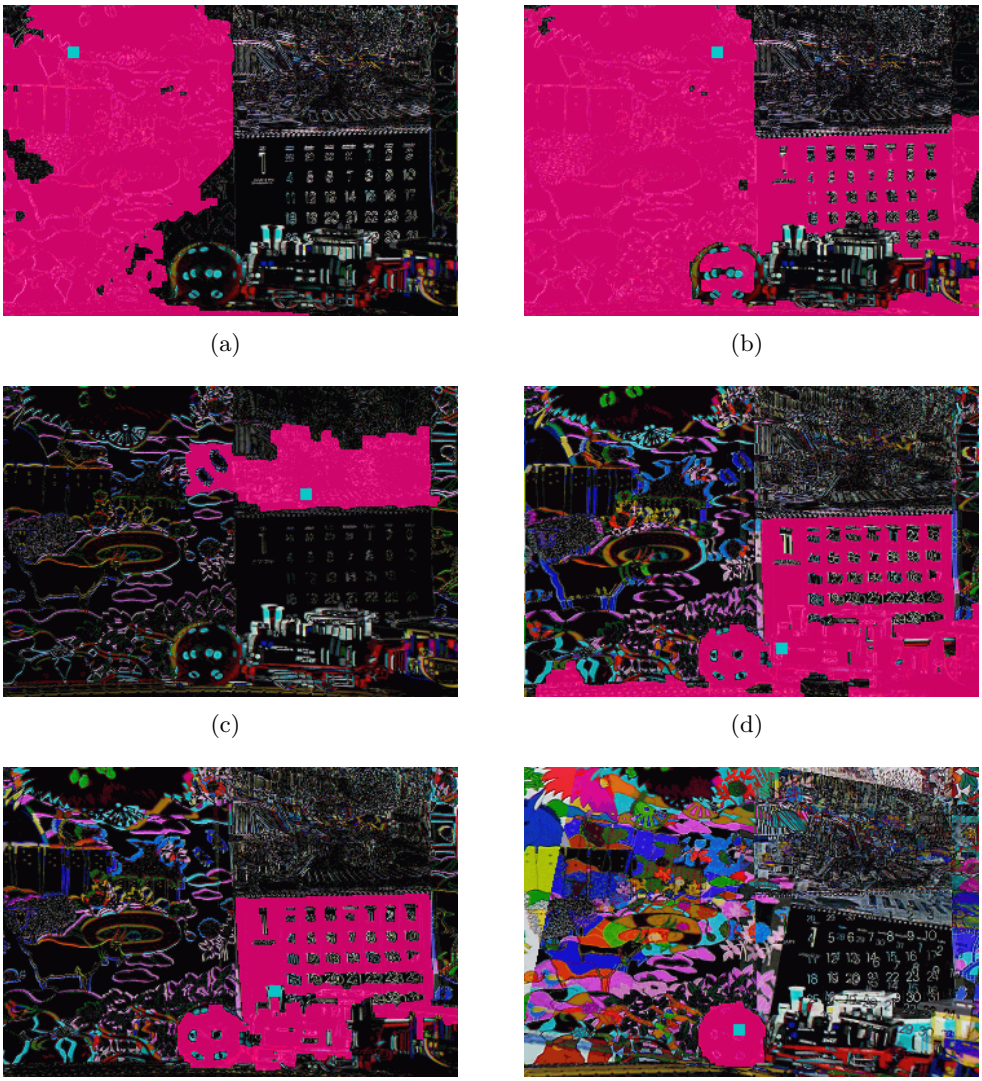
Abbildung 5.5: Region Growing eines Saatpunktes im Kalender. In (a) bis (b) sind die affinen Parameter des Kalenders gut bestimmt. In (b) sieht man schon, wie die Region an einer schmalen Stelle in den Hintergrund hineinwächst. In (c) unterstützen die affinen Parameter auch den Hintergrund, man erkennt, dass der Hintergrund schwarz wird. In (d) ist der Bereich außerhalb des Kalenders größer als innerhalb. In (e) wächst die Region in den unteren Bereich des Kalenders. Das Ergebnis des Region Growing für diesen Saatpunkt ist in (f) dargestellt.

```
    x0     y0     x1     y1     axx     ayx     axy     ayy  Groesse
( 54.0,  36.0,  56.9,  36.9,  0.992,  0.000,  0.000,  0.988) [36065]
(149.0,  36.0, 151.0,  37.0,  0.989,  0.001, -0.001,  0.988) [55495]
(218.0,  78.0, 219.2,  79.8,  0.987,  0.001,  0.000,  0.987) [9126]
(234.0,  83.0, 235.0,  84.8,  0.987,  0.001,  0.000,  0.987) [8946]
(199.0, 202.0, 195.5, 200.9,  0.992, -0.003, -0.004,  0.982) [21078]
(210.0, 173.0, 207.8, 172.6,  0.978, -0.005, -0.054,  0.973) [15187]
(166.0, 203.0, 161.2, 201.2,  0.965, -0.168,  0.201,  0.969) [2130]
```

Abbildung 5.6: Kommandozeilenausgabe nach dem Region Merging Schritt. Sehr ähnliche affine Bewegungen wurden durch den Region Merging Schritt zusammengefügt, sodass aus ursprünglich 43 Bewegungen nur noch 7 übrig geblieben sind.

## 5.3 Region Merging Ergebnis

Mit der Initialisierung, wie sie in der Beispieldatei 4.1 angegeben wurde, werden vom Region Growing Schritt insgesamt 43 Regionen mit dazugehörigen affinen Parametern bestimmt. Der Region Merging Schritt führt ähnliche Regionen zusammen und bestimmt für die zusammengeführten Regionen die affinen Parameter neu. So reduziert der Region Merging Schritt die ursprünglich 43 zu nur mehr 7 affinen Parametern. Die Kommandozeilenausgabe in Abbildung 5.6 zeigt die resultierenden 7 affinen Parameter des Region Mergings. Die dazugehörigen Regionen sieht man in Abbildung 5.7. Die Kommandozeilenausgabe aus dem Region Growing Schritt, mit den 43 affinen Parametern, ist in der Kommandozeilenausgabe in Abbildung 5.8 dargestellt. Die Ziffer ganz rechts in dieser Abbildung informiert, welche affinen Parameter der Merging Schritt zusammenführt.



(a) (b) (c) (d) (e) (f)

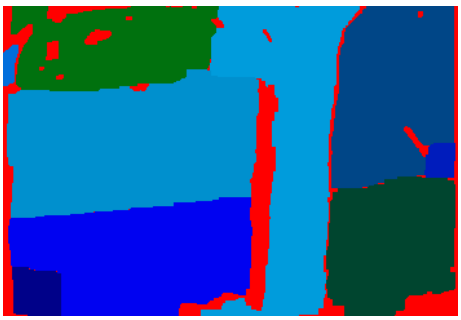
Abbildung 5.7: Ergebnis des Region Merging. Nach dem Zusammenführen sind 7 affine Parameter übrig geblieben. Die Kalenderregion (c) ist fast identisch ein zweites Mal bestimmt worden und wird hier nicht abgebildet. Die affinen Parameter in (e) unterstützen weder den Kalender noch den Zug richtig. (b) unterstützt den Hintergrund besser als (a), das erkennt man gut am Hintergrund rechts vom Kalender.

```
    x0     y0     x1     y1     axx     ayx     axy     ayy  Groesse
( 50.0, 199.0,  53.1, 197.9,  0.994,  0.000,  0.000,  0.989) [37010] 2
( 93.0, 203.0,  95.5, 201.9,  0.989,  0.001, -0.001,  0.988) [54644] 2
( 71.0, 203.0,  73.8, 201.9,  0.989,  0.001, -0.001,  0.988) [53946] 2
( 72.0, 111.0,  74.8, 111.0,  0.989,  0.001, -0.001,  0.988) [54604] 2
( 85.0, 177.0,  87.6, 176.3,  0.989,  0.001, -0.001,  0.988) [54133] 2
( 89.0, 187.0,  91.6, 186.1,  0.989,  0.001, -0.001,  0.988) [54153] 2
( 61.0, 169.0,  63.9, 168.3,  0.989,  0.001, -0.001,  0.988) [54121] 2
( 73.0, 179.0,  75.8, 178.2,  0.989,  0.001, -0.001,  0.988) [54135] 2
(169.0, 117.0, 170.7, 117.0,  0.989,  0.001, -0.001,  0.988) [54676] 2
( 77.0,  52.0,  79.8,  52.7,  0.989,  0.001, -0.001,  0.988) [55083] 2
( 43.0,  41.0,  46.2,  41.8,  0.989,  0.001, -0.001,  0.988) [55082] 2
( 58.0, 113.0,  61.0, 113.0,  0.989,  0.001, -0.001,  0.988) [54117] 2
( 94.0, 142.0,  96.5, 141.7,  0.989,  0.001, -0.001,  0.988) [54101] 2
(141.0, 131.0, 143.0, 130.8,  0.989,  0.001, -0.001,  0.988) [55031] 2
(142.0,  86.0, 144.0,  86.4,  0.989,  0.001, -0.001,  0.988) [53512] 2
(139.0, 120.0, 141.0, 120.0,  0.989,  0.001, -0.001,  0.988) [53518] 2
(149.0, 102.0, 150.9, 102.2,  0.989,  0.001, -0.001,  0.988) [53513] 2
(148.0, 141.0, 149.9, 140.7,  0.989,  0.001, -0.001,  0.988) [53498] 2
(139.0, 102.0, 141.1, 102.2,  0.989,  0.001, -0.001,  0.988) [55034] 2
(149.0, 112.0, 150.9, 112.1,  0.989,  0.001, -0.001,  0.988) [53522] 2
(127.0, 155.0, 129.2, 154.5,  0.989,  0.001, -0.001,  0.988) [53501] 2
(151.0, 131.0, 152.9, 130.8,  0.989,  0.001, -0.001,  0.988) [53502] 2
(129.0, 116.0, 131.2, 116.0,  0.989,  0.001, -0.001,  0.988) [54021] 2
(147.0, 175.0, 148.9, 174.3,  0.989,  0.001, -0.001,  0.988) [55016] 2
(157.0, 172.0, 158.8, 171.4,  0.989,  0.001, -0.001,  0.988) [53477] 2
(128.0, 181.0, 130.1, 180.2,  0.989,  0.001, -0.001,  0.988) [54078] 2
(143.0, 165.0, 145.0, 164.4,  0.989,  0.001, -0.001,  0.988) [55052] 2
(113.0, 192.0, 115.3, 191.1,  0.989,  0.001, -0.001,  0.988) [54054] 2
(105.0,  65.0, 107.5,  65.6,  0.989,  0.001, -0.001,  0.988) [54407] 2
(149.0,  36.0, 151.0,  37.0,  0.989,  0.001, -0.001,  0.988) [55284] 2
(174.0,  39.0, 175.7,  40.0,  0.989,  0.001, -0.001,  0.988) [54896] 2
(139.0,  40.0, 141.1,  40.9,  0.989,  0.001, -0.001,  0.988) [54629] 2
( 36.0,  51.0,  39.1,  51.8,  0.990, -0.000, -0.001,  0.988) [31160] 2
( 54.0,  36.0,  57.0,  36.9,  0.990, -0.000, -0.001,  0.988) [27790] 1
( 40.0, 199.0,  43.2, 197.9,  0.998,  0.003,  0.001,  0.987) [30986] 1
(218.0,  78.0, 219.2,  79.8,  0.987,  0.001,  0.000,  0.987) [9116]  3
(244.0,  84.0, 244.8,  85.8,  0.987,  0.001,  0.000,  0.987) [9098]  3
(234.0,  83.0, 235.0,  84.8,  0.987,  0.001,  0.000,  0.987) [8946]  4
(235.0, 190.0, 231.3, 189.0,  0.991, -0.003,  0.001,  0.983) [20612] 5
(225.0, 185.0, 221.4, 184.1,  0.992, -0.003, -0.001,  0.983) [20760] 5
(199.0, 202.0, 195.5, 200.9,  0.992, -0.003, -0.004,  0.982) [20898] 5
(210.0, 173.0, 207.8, 172.6,  0.978, -0.005, -0.054,  0.973) [15187] 6
(166.0, 203.0, 161.2, 201.2,  0.965, -0.168,  0.201,  0.969) [2130]  7
```

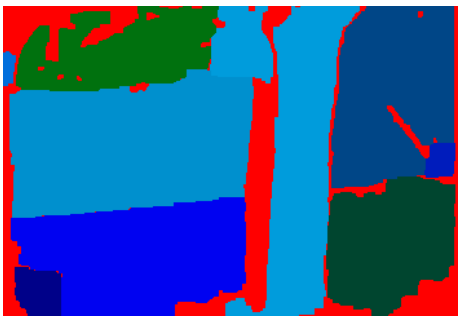
Abbildung 5.8: Kommandozeilenausgabe nach dem Region Growing Schritt. Insgesamt wurden 43 affine Parameter gefunden. Die Zahl ganz rechts ist kein Teil der Ausgabe. Sie wurde im Nachhinein dazugefügt und kennzeichnet, welche Regionen der folgende Region Merging Schritt zusammenführen wird.

(a) Bildpaar $\{1, 2\}$

(b) Bildpaar $\{1, 3\}$



(c) Bildpaar $\{1, 4\}$



(d) Bildpaar $\{1, 5\}$

Abbildung 5.9: Zuordnungen bei den Bildpaaren einer Eingabesequenz mit fünf Bildern mit den Parametern $occlusion = 10.0$ und $lambdaL = 30.0$. Ein Bildpunkt muss zwischen den Bildpaarzuordnungen entweder seinen ursprünglichen Label behalten oder verdeckt werden. Je größer der Abstand der Bilder innerhalb eines Bildpaares wird, desto mehr Bildpunkte sind verdeckt. Man erkennt eine Verdeckungszunahme von der Zuordnung in (a) bis zur Zuordnung in (d).

## 5.4 Layer Assignment Ergebnis

Beim Zuordnungsschritt sind die Parameter *lambdaL* und der Parameter *occlusion* festzulegen. Im Folgenden werden die Ausgaben des Programms dargestellt, die durch unterschiedliche Parameter und Eingabesequenzen entstehen. Bei einer Zuordnung sind immer alle Eingabebilder betroffen. Das erste Bild der Eingabesequenz ist immer das Referenzbild, und alle weiteren Bilder werden mit diesem in Beziehung gesetzt. Das Ergebnis des Zuordnungsschritts ist die Zuordnung der Bildpunkte zwischen dem Bildpaar $\{1, 2\}$, dem Bildpaar $\{1, 3\}$, und so weiter. Der einzige Unterschied zwischen den Bildpaarzuordnungen ist, dass die Verdeckungen zunehmen, je größer der Abstand der Bilder innerhalb des Bildpaares wird. Abbildung 5.9 zeigt das anschaulich. Da sich die Bildpaarzuordnungen nur in den Verdeckungen unterscheiden, wird in den kommenden Beispielen nur die Zuordnung zwischen dem Bildpaar $\{1, 2\}$ gezeigt. Um die Auswirkung des Parameters *lambdaL* zu untersuchen, werden zuerst die Verdeckungskosten sehr groß

eingestellt und dann der Parameter *lambdaL* sukzessive von sehr klein bis sehr groß variiert. In Abbildung 5.10 sieht man, was für einen Einfluss der Parameter *lambdaL* auf das Ergebnis hat. Um die Auswirkungen des Parameters *occlusion* zu untersuchen, werden die Smoothnesskosten festgesetzt und dann der Parameter *occlusion* sukzessive von sehr klein bis groß variiert. In Abbildung 5.11 sieht man, was für einen Einfluss der Parameter *occlusion* auf das Ergebnis hat.

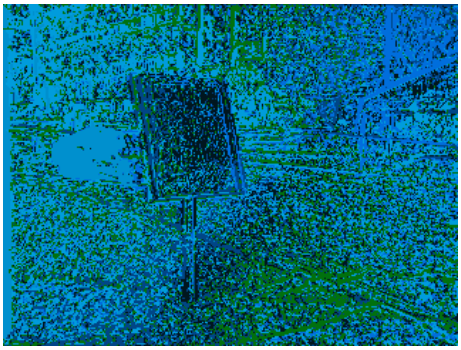
Leider lässt sich die Qualität der Ergebnisse nicht objektiv feststellen, da für keine Eingabesequenz ein Ground Truth vorhanden ist, der jedem Bildpunkt einen Bewegungsvektor zuordnet. Darum bleibt die Beurteilung der Ergebnisse in dieser Arbeit eine subjektive. Es ist leicht möglich, dass der menschliche Betrachter einen Hang zum Oversmoothen hat. Beispielsweise nimmt der menschliche Betrachter einen Würfel als ein Objekt war. Bewegt sich der Würfel, schreibt ihm der Mensch eine Bewegung zu, obwohl jede Würfelseite eine andere Bewegung durchführt.

Auf den nächsten Seiten sind Programmausgaben, die mit mehreren Testsequenzen gemacht wurden. Die Parameter wurden dabei so angepasst, dass subjektiv gute Ergebnisse erzielt wurden. Die Parameter für die einzelnen Testsequenzen sind in der Tabelle 5.1 dargestellt. Die Programmausgaben auf den folgenden Seiten beinhalten jeweils: Das Referenzbild der Testsequenz, eine identifizierte Region aus dem Region Growing Schritt, das Assignment Ergebnis Bild, einen Quiver Plot und zwei Graustufenbilder. Das Referenzbild ist immer das erste Bild der Eingabesequenz, das Assignment Ergebnis Bild bezieht sich auf dieses Bild. Die Graustufenbilder drücken die absolute Bewegung in X Richtung bzw. in Y Richtung aus. Ein schwarzer Bildpunkt entspricht keiner Bewegung, ein weißer Bildpunkt entspricht einer absoluten Bewegung, die mindestens 3 Prozent der Bildbreite entspricht. Der Quiver Plot stellt die Bewegungsvektoren als Linien dar. Das Linienende mit der Serife ist der Startpunkt des Vektors.

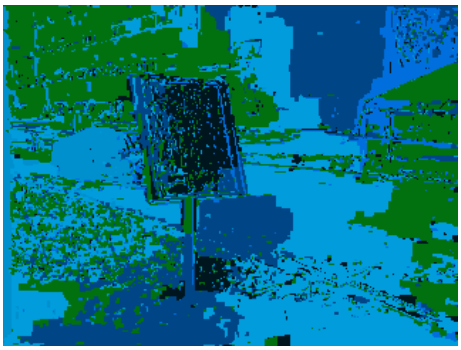
Die Ergebnisse von Jiangjan Xiao und Mubarak Shah zeigen etwas exaktere Objektgrenzen bei der Zuordnung. Man erkennt das zum Beispiel am Zug Layer in der *Mobile & Calendar* Sequenz. Bei J. Xiao und M. Shah wird der Waggon vom Kalenderhintergrund getrennt. In dieser Arbeit wird ein Teil des Kalenders zum Waggon dazugenommen. Es ist am Kalenderhintergrund ein Schatten des Zuges zu sehen, der sich natürlich mit dem Zug mitbewegt und deshalb als Teil des Zuges wahrgenommen werden kann, dennoch gehört dieser Bereich zum Kalender. Die Unterschiede zu den Ergebnissen von J. Xiao und M. Shah liegen wahrscheinlich in den Anpassungen – siehe Kapitel 3.3.3 –, die in dieser Arbeit gemacht wurden. Eventuell ließen sich andere Anpassungen finden, um den Ergebnissen von J. Xiao und M. Shah näher zu kommen, denn gerade die Smoothnesskosten wurden in dieser Arbeit stark vereinfacht, um die Energiefunktion im Flussnetzwerk abbilden zu können. Im Folgenden wird auf die Ergebnisse im Einzelnen eingegangen.

**Moving Car** Die Ergebnisse dieser Sequenz sind in Abbildung 5.12 dargestellt. Nach dem Region Merging Schritt bleiben 6 affine Bewegungen übrig. Der Layer Assignment Schritt erkennt das Auto sehr schön. Der Schatten vom Auto auf der Straße wird zum Auto gezählt. Automatisch, also ohne das Erfahrungswissen eines Menschen, scheint es überhaupt nicht möglich zu sein, den Schatten nicht zum Auto zu zählen.

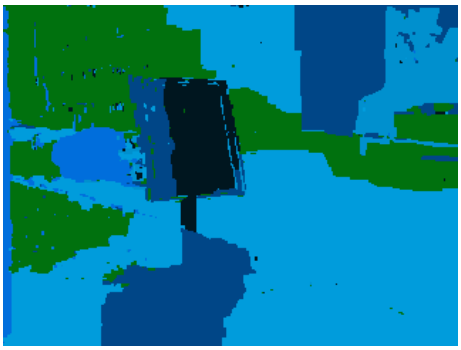
(a) Referenzbild



(b) Zuordnung bei $lambdaL = 0.0$



(c) Zuordnung bei $lambdaL = 0.1$



(d) Zuordnung bei $lambdaL = 1.0$

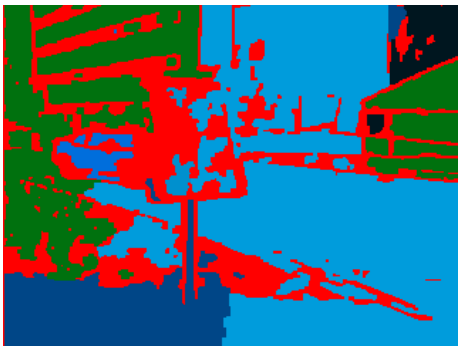
(e) Zuordnung bei $lambdaL = 10.0$

(f) Zuordnung bei $lambdaL = 100.0$

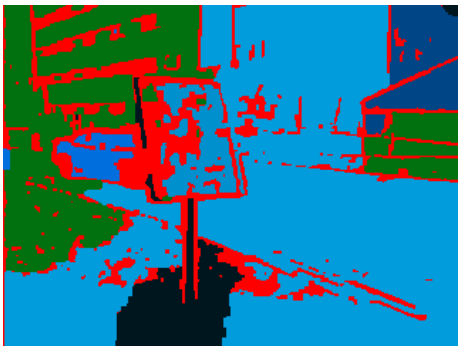
Abbildung 5.10: Unterschiedliche Werte für den Smoothnessparameter *lambdaL*. Der Parameter *occlusion* hatte immer den Wert 500.0.

(a) Referenzbild

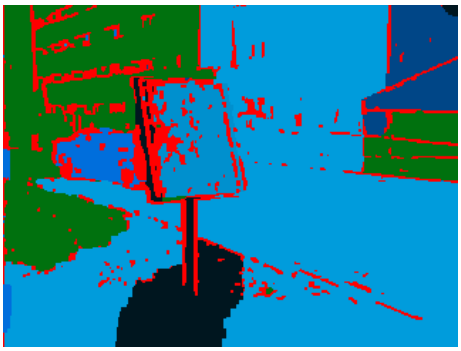
(b) Zuordnung bei *occlusion* = 2.0



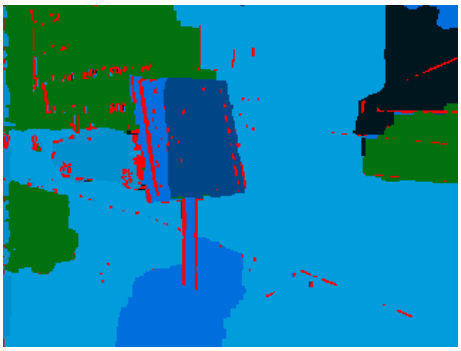
(c) Zuordnung bei *occlusion* = 2.5



(d) Zuordnung bei *occlusion* = 3.0



(e) Zuordnung bei *occlusion* = 4.0



(f) Zuordnung bei *occlusion* = 8.0

Abbildung 5.11: Unterschiedliche Werte für den Verdeckungsparameter *occlusion*. Der Parameter *lambdaL* hatte immer den Wert 2.0. Die roten Bereiche markieren die verdeckten Bildpunkte.

Tabelle 5.1: Mit diesen Parametern erzeugt das Programm Lay jene Ausgaben, die in dieser Arbeit dargestellt sind. Jede Zeile stellt einen Parameter dar. Die Spalten entsprechen den folgenden Testsequenzen: a der Testsequenz *Moving Car*, b der Testsequenz *Mobile & Calendar*, mit Startindex 0, c der Testsequenz *Mobile & Calendar*, mit Startindex 40, d der Testsequenz *Flower Garden*, e der Testsequenz *Tennis*, f der Testsequenz *Selfrecorded Train*, g der Testsequenz *Sequenz 2*, h der Testsequenz *Sequenz 3* und i der Testsequenz *Sequenz 6*.

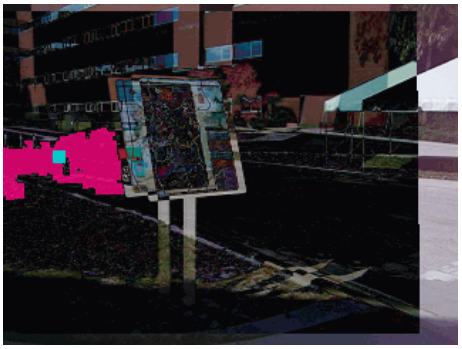
| Sequenz | a | b | c | d | e | f | g | h | i |
|---|---|---|---|---|---|---|---|---|---|
| numberofimages | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 4 |
| startindex | 1 | 0 | 40 | 0 | 0 | 5 | 0 | 0 | 1 |
| featureanz | 50 | 100 | 100 | 150 | 70 | 100 | 100 | 100 | 100 |
| windowsize_trans | 9 | 11 | 11 | 9 | 9 | 9 | 9 | 9 | 7 |
| windowsize | 15 | 19 | 19 | 15 | 15 | 15 | 19 | 15 | 15 |
| grad_sigma | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 |
| smooth_sigma_fact | 0.1 | 0.1 | 0.1 | 0.15 | 0.1 | 0.1 | 0.1 | 0.1 | 0.15 |
| lambdaF | 8.0 | 22.0 | 8.0 | 3.5 | 10.0 | 15.0 | 6.0 | 15.0 | 8.0 |
| threshold | 11.5 | 20.0 | 19.0 | 15.0 | 10.0 | 25.0 | 16.0 | 15.0 | 14.0 |
| fwhm | 5.0 | 5.0 | 5.0 | 7.0 | 10.0 | 5.0 | 5.0 | 5.0 | 5.0 |
| gaussborder | 3.0 | 6.0 | 6.0 | 6.0 | 10.0 | 6.0 | 6.0 | 6.0 | 6.0 |
| lambdaL | 16.0 | 30.0 | 30.0 | 30.0 | 10.0 | 15.0 | 52.0 | 30.0 | 25.0 |
| occlusion | 20.0 | 15.0 | 20.0 | 20.0 | 10.0 | 30.0 | 26.0 | 15.0 | 20.0 |

Bei dieser Sequenz gab es die Schwierigkeit, die Parameter so zu justieren, dass die Tafel mit der Stadtkarte neben der Straße im Region Growing Schritt erkannt wird. Waren die Parameter zu tolerant eingestellt, dann ist die Region über die Tafel hinausgewachsen und hat zu sehr die affine Bewegung des Hintergrunds angenommen. Waren die Parameter zu begrenzend eingestellt, dann ist die Region, wegen der starken Textur in der Landkarte, überhaupt nicht gewachsen.

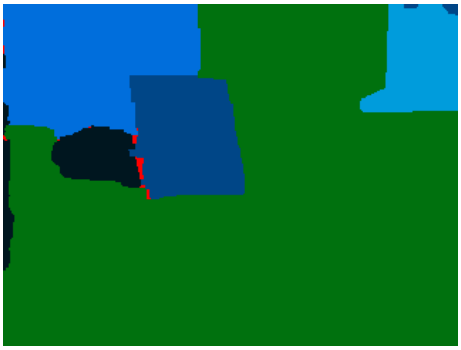
Der Stiel der Stadtkarte wurde nicht erkannt und dem Hintergrund zugeordnet. Für den Algorithmus, der ausschließlich die Bewegung als Kriterium verwendet, ist es schwer den Stiel gegen den Hintergrund zu trennen, da der Abstand zwischen Stiel und Hintergrund klein ist. Der Abstand zwischen Stadtkarte und Hintergrund ist wesentlich größer.

**Mobile & Calendar, mit Startindex 0** Die Ergebnisse dieser Sequenz sind in Abbildung 5.13 dargestellt. Nach dem Region Merging Schritt bleiben 7 affine Bewegungen übrig. Von diesen werden 4, im Layer Assignment Schritt, den Bildpunkten zugeordnet. Der Kalender wird vom Layer Assignment Schritt ausgezeichnet erkannt. Der Ball Layer wird nicht exakt bestimmt. Hier werden einige Bildpunkte des Hintergrunds dazugenommen. Die Spielzeugeisenbahn wird ganz gut erkannt. Hier gibt es dieselbe Schwierigkeit wie bei der *Moving Car* Sequenz, dass der Schatten der Eisenbahn dem Eisenbahn Layer zugeordnet wird und nicht dem Kalender Layer.

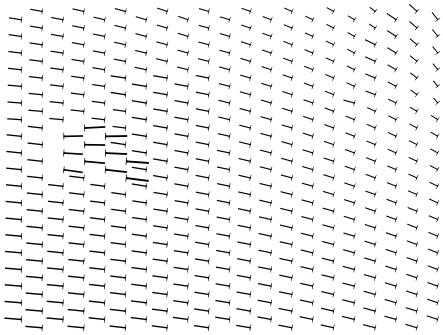
(a) Referenzbild



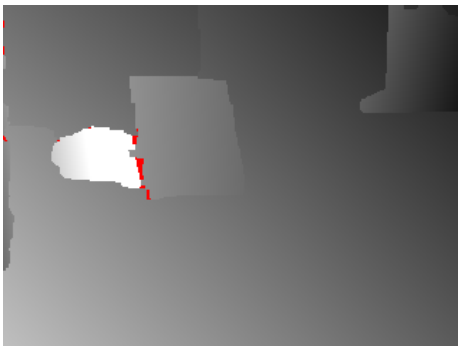
(b) Eine identifizierte Bewegung



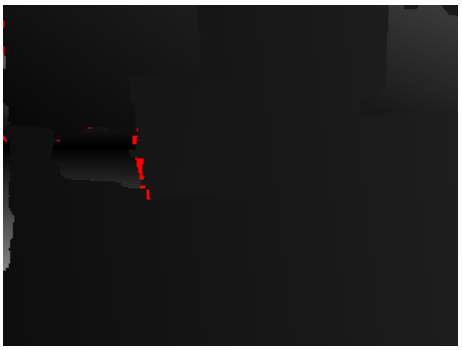
(c) Zuordnung



(d) Quiver Bild



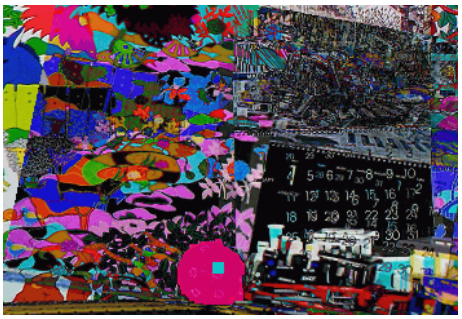
(e) Bewegung in X Richtung



(f) Bewegung in Y Richtung

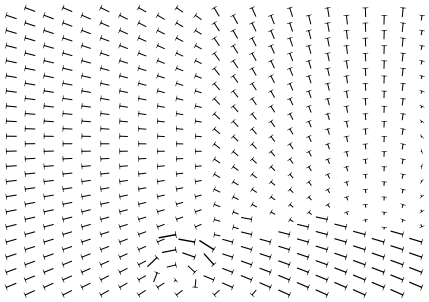
Abbildung 5.12: Ergebnis für die Eingabesequenz *car*.

(a) Referenzbild

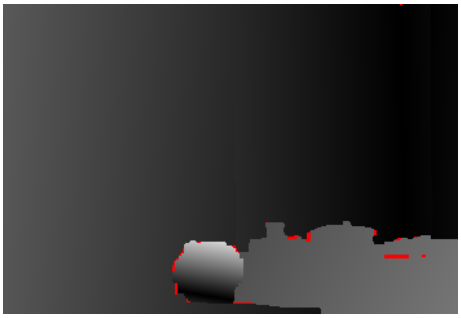


(b) Eine identifizierte Bewegung

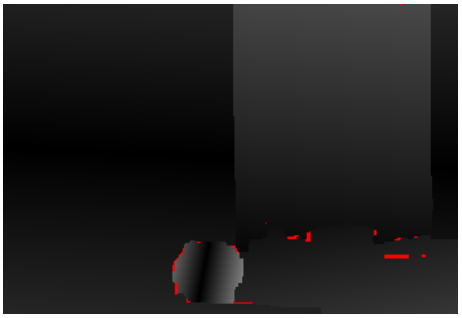
(c) Zuordnung



(d) Quiver Bild



(e) Bewegung in X Richtung



(f) Bewegung in Y Richtung

Abbildung 5.13: Ergebnis für die Eingabesequenz *Mobile & Calendar*.

**Mobile & Calendar, mit Startindex 40** Die Ergebnisse dieser Sequenz sind in Abbildung 5.14 dargestellt. Nach dem Region Merging Schritt bleiben 8 affine Bewegungen übrig. Die Ball und Kalender Layer werden sehr gut erkannt. Die Eisenbahn mit dem Waggon wird nicht besonders gut abgegrenzt, hier werden zu viele Bereiche des Hintergrunds zum Lokomotiven Layer dazugenommen.

**Flower Garden** Die Ergebnisse dieser Sequenz sind in Abbildung 5.15 dargestellt. Nach dem Region Merging Schritt bleiben 36 affine Bewegungen übrig. Der Baum wird recht gut durch zwei affine Bewegungen beschrieben, und die Objektgrenzen vom Baumstamm sind ziemlich exakt. Die Baumkrone enthält zu viele Bereiche des Himmel Hintergrunds, der wegen der spärlichen Textur schwierig zu trennen ist. Die Hauptbewegungsbereiche – Baum, Blumenwiese, Häuser einschließlich Himmel – sind an den richtigen Stellen voneinander getrennt worden. Am linken Baumstammrand treten, wie erwartet, die meisten verdeckten Bereiche auf. Am äußerst rechten Bildrand ist in der Eingabesequenz ein schwarzer Streifen von zwei Bildpunkten, der nichts mit der Szene zu tun hat. Man kann ihn trotzdem als künstliches Objekt betrachten, bei dem das *Aperture Problem* – siehe Kapitel 1.2 – bei der Zuordnung auftrat.

**Tennis** Die Ergebnisse dieser Sequenz sind in Abbildung 5.16 dargestellt. Nach dem Region Merging Schritt bleiben 12 affine Bewegungen übrig. Die affinen Bewegungen werden, oberflächlich betrachtet, richtig zugeordnet. Die Objekte werden jedoch nicht exakt abgegrenzt. So wurden kleine Bereiche der Hand des Spielers, als auch des Tischtennisschlägers, dem Hintergrund zugeordnet. Andererseits wurden Bereiche des Hintergrunds dem Ball Layer und dem Spieler Arm Layer zugeordnet. Die verdeckten Bereiche sind zum Teil fragwürdig.

**Selfrecorded Train** Die Ergebnisse dieser Sequenz sind in Abbildung 5.17 dargestellt. Nach dem Region Merging Schritt bleiben 19 affine Bewegungen übrig, von denen aber viele sehr ähnlich sind. Der Layer Assignment Schritt ordnet den Bildpunkten auch nur 7 von den 19 affinen Bewegungen zu. Wenn der Region Merging Schritt etwas toleranter wäre, könnten die 19 Bewegungen bestimmt noch um die Hälfte verkleinert werden. Das würde den Assignment Schritt beschleunigen.

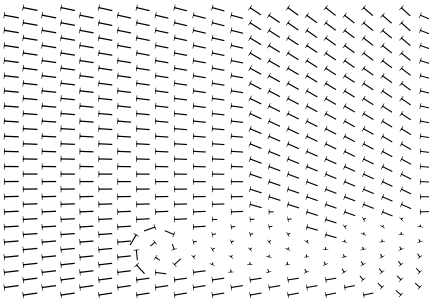
Man erkennt am Quiver Plot sehr schön, wie das stark verzerrende Objektiv der Kamera aus der geradlinigen Bewegung des Zuges eine gekrümmte macht. Das spricht für die Qualität des Region Growing Schrittes, der diese affinen Bewegungen findet.

Die Objekte werden zum Teil falsch abgegrenzt. Das lag zum Teil am lokal texturlosen Hintergrund. So wurden zum Beispiel Bereiche des Buch Hintergrunds – Statistical Pattern Recognition – dem Zug Layer zugeordnet. Außerdem trat das *Aperture Problem* wieder auf, das sich zum Beispiel daran zeigt, dass Bereiche der Schienen dem Zug Layer zugeordnet wurden. Ob sich die Schiene mit dem Zug bewegt oder still steht, kann aus einer lokalen Sicht, bei der kein Ende eines Schienenstranges sichtbar ist, nicht beurteilt werden. Außerdem wurde ein Bereich am unteren Bildrand dem Zug Layer zugeordnet. Auch diese Zuordnung kann dem Aperture Problem zugeschrieben werden.

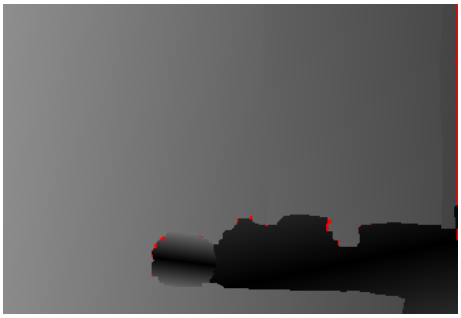
(a) Referenzbild

(b) Eine identifizierte Bewegung

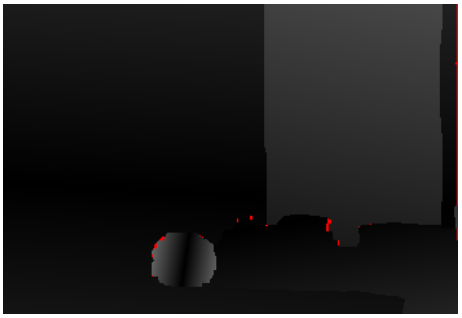
(c) Zuordnung



(d) Quiver Bild



(e) Bewegung in X Richtung



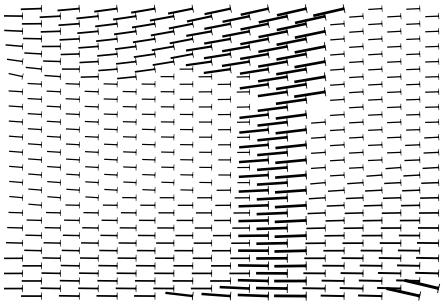
(f) Bewegung in Y Richtung

Abbildung 5.14: Ergebnis für die Eingabesequenz *Mobile & Calendar*.

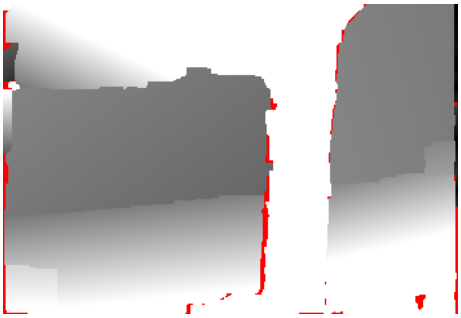
(a) Referenzbild

(b) Eine identifizierte Bewegung

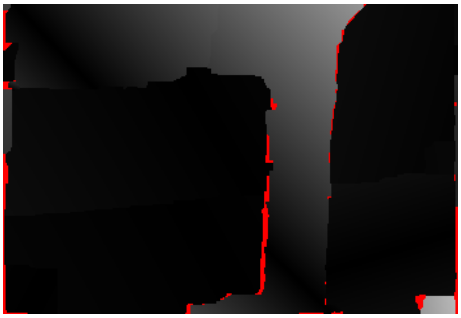
(c) Zuordnung



(d) Quiver Bild



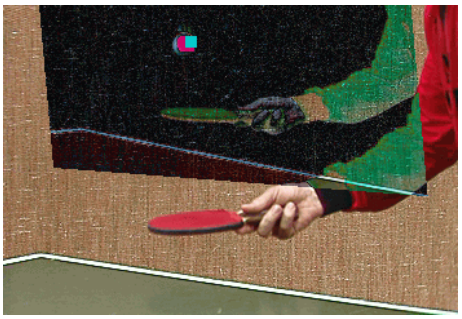
(e) Bewegung in X Richtung



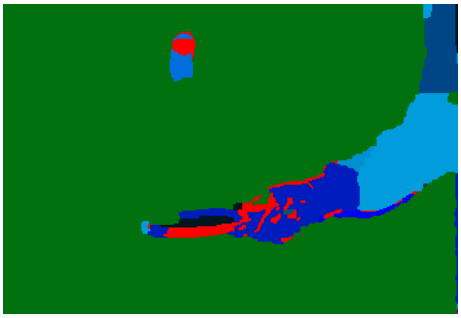
(f) Bewegung in Y Richtung

Abbildung 5.15: Ergebnis für die Eingabesequenz *Flower Garden*.

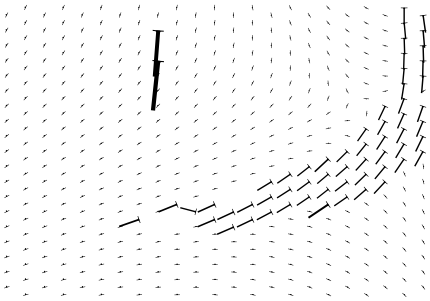
(a) Referenzbild



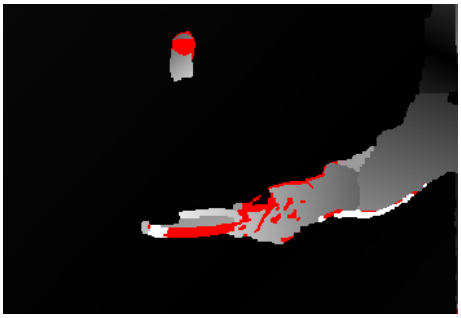
(b) Eine identifizierte Bewegung



(c) Zuordnung



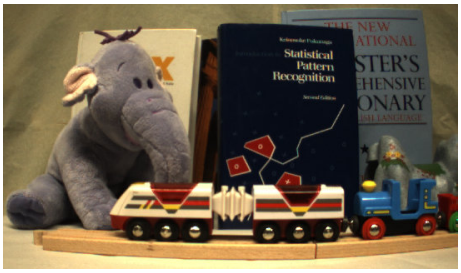
(d) Quiver Bild



(e) Bewegung in X Richtung

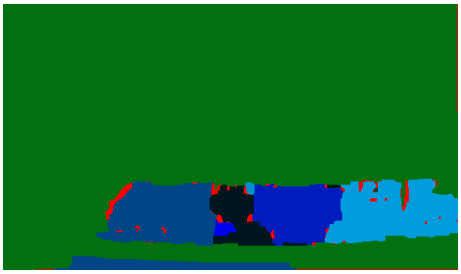
(f) Bewegung in Y Richtung

Abbildung 5.16: Ergebnis für die Eingabesequenz *Tennis*.

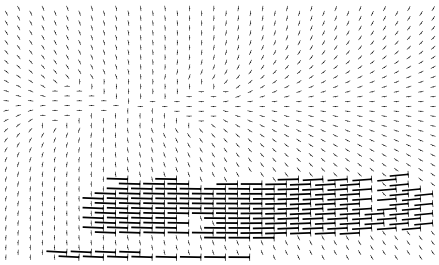



(a) Referenzbild

(b) Eine identifizierte Bewegung

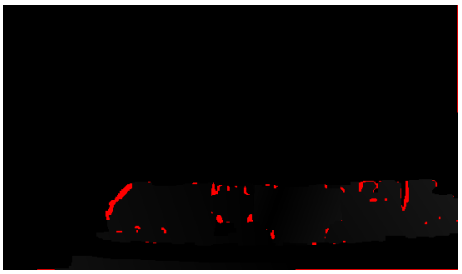


(c) Zuordnung



(d) Quiver Bild

(e) Bewegung in X Richtung



(f) Bewegung in Y Richtung

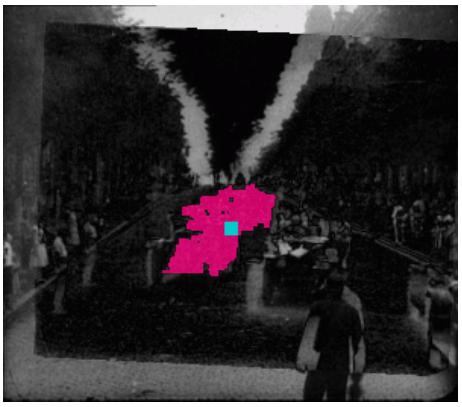
Abbildung 5.17: Ergebnis für die Eingabesequenz *Zug*.

**Sequenz 2** Die Ergebnisse dieser Sequenz sind in Abbildung 5.18 dargestellt. Nach dem Region Merging Schritt bleiben 16 affine Bewegungen übrig. Die affinen Bewegungen der beiden Fahrzeuge wurden gut erkannt. Man erkennt im Quiver Plot durch die nach links unten weisenden und gleichzeitig auseinanderstrebenden Pfeile, wie die Fahrzeuge der Kamera entgegenfahren und dadurch immer größer werden. Bereiche des Hintergrunds, welche verdeckt sein sollten, werden den Fahrzeug Layern zugeordnet. Der Person hinter dem rechten Fahrzeug wird korrekterweise der Verdeckungslabel zugeordnet. Die Person im Vordergrund wird, bis auf den fehlenden Kopf, gut gegen den Hintergrund abgegrenzt.

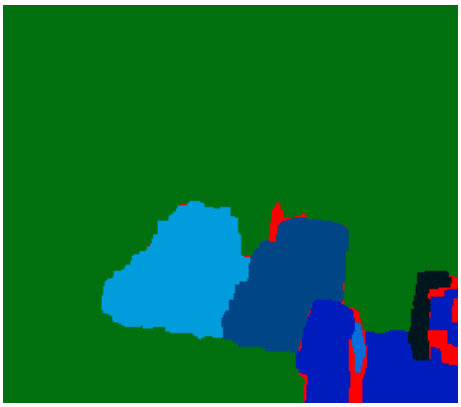
**Sequenz 3** Die Ergebnisse dieser Sequenz sind in Abbildung 5.19 dargestellt. Nach dem Region Merging Schritt bleiben 13 affine Bewegungen übrig. Der zur Kamera fahrende Zug erzeugt einen ähnlichen Quiver Plot wie die Fahrzeuge in Sequenz 2. Die Pfeile im Plot werden immer länger und streben auseinander. Die Abgrenzung der Waggons auf der linken Seite ist nicht genau. Das liegt vermutlich daran, dass diese Seite des Zugs im Schatten liegt und sich dieser mit dem Zug bewegt.

**Sequenz 6** Die Ergebnisse dieser Sequenz sind in Abbildung 5.20 dargestellt. Nach dem Region Merging Schritt bleiben 11 affine Bewegungen übrig. Der Layer Assignment Schritt hat hier, trotz der vermeintlich einfachen Szenenbeschreibung, einige Schwierigkeiten, die Bildpunkte korrekt zuzuordnen. Der Bereich zwischen den Waggons, der zum Hintergrund gehört, wird wegen Texturlosigkeit zum Waggon Layer dazugezählt. Der Bereich des Waggons, der aus dem Bild verschwindet, sollte verdeckt sein, bekommt aber den Hintergrund Layer zugeordnet. Ein Bereich in der Brücke bekommt eine Bewegung zugeordnet, wo keine sein sollte. Im Bereich der Räder gibt es einige Verdeckungen. Das kommt daher, dass die Räder Speichen haben und es dadurch zu Unstetigkeiten in diesen Bereichen kommt.

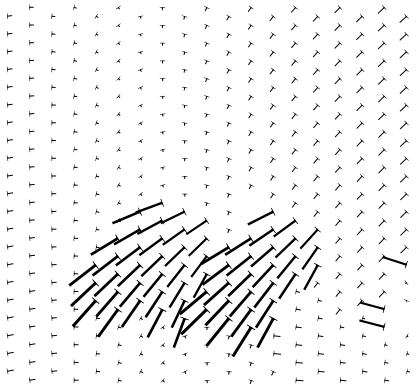
(a) Referenzbild



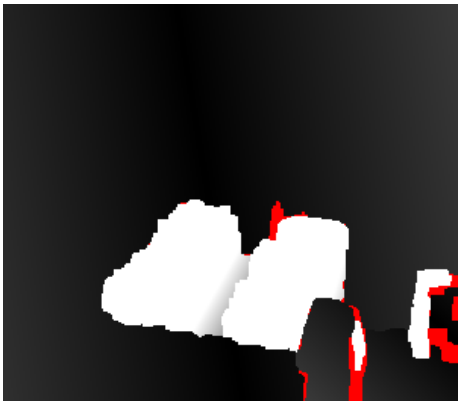
(b) Eine identifizierte Bewegung



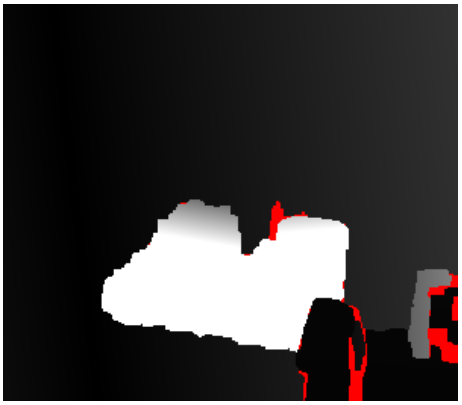
(c) Zuordnung



(d) Quiver Bild



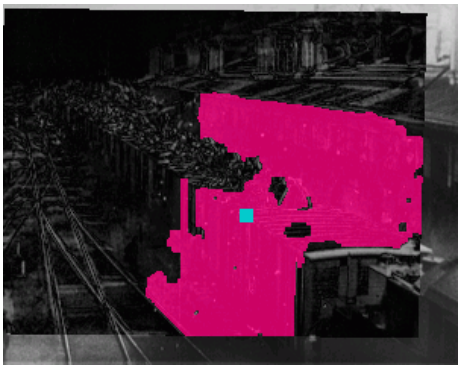
(e) Bewegung in X Richtung



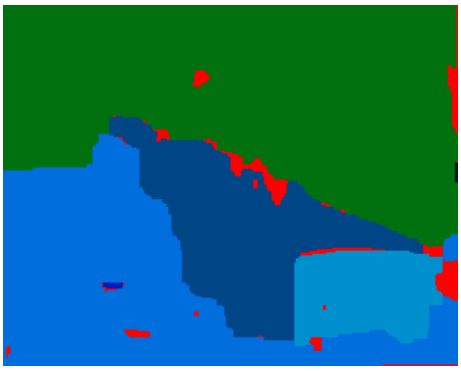
(f) Bewegung in Y Richtung

Abbildung 5.18: Ergebnis für die Eingabesequenz *Sequenz 2*.

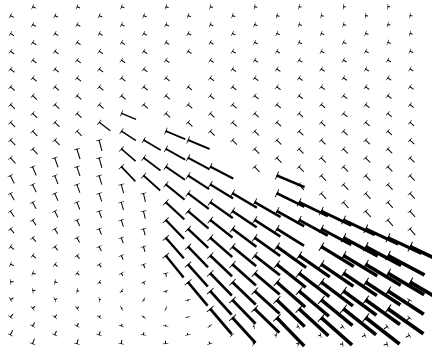
(a) Referenzbild



(b) Eine identifizierte Bewegung

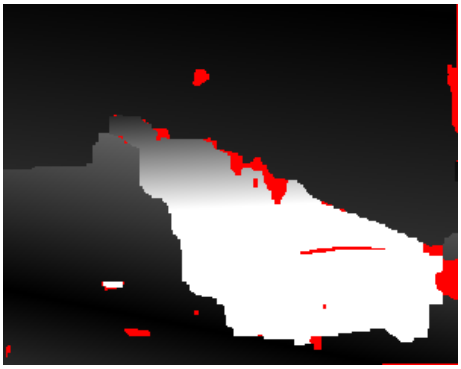


(c) Zuordnung



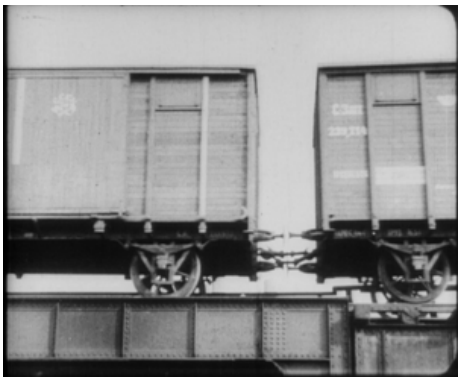
(d) Quiver Bild

(e) Bewegung in X Richtung

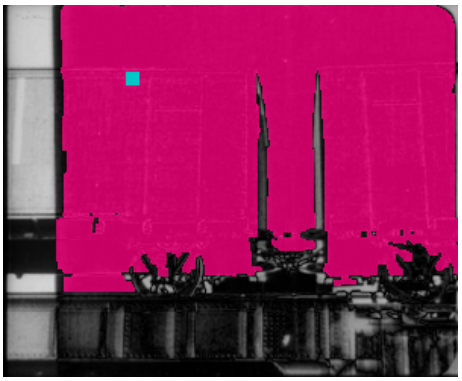


(f) Bewegung in Y Richtung

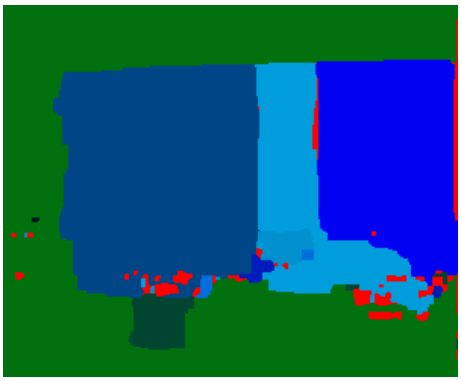
Abbildung 5.19: Ergebnis für die Eingabesequenz *Sequenz 3*.



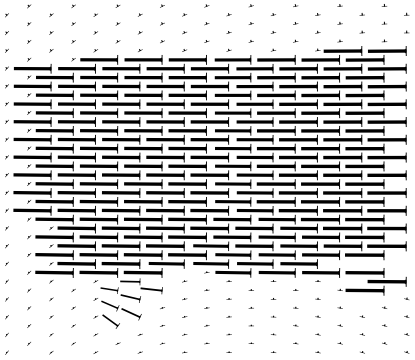
(a) Referenzbild



(b) Eine identifizierte Bewegung

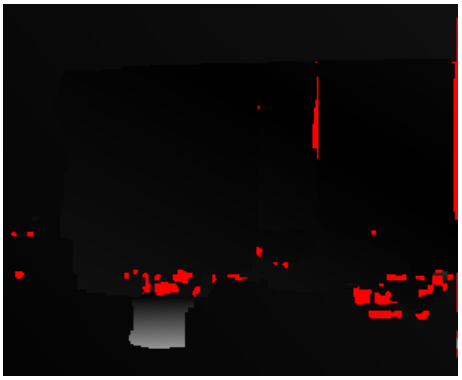


(c) Zuordnung



(d) Quiver Bild

(e) Bewegung in X Richtung



(f) Bewegung in Y Richtung

Abbildung 5.20: Ergebnis für die Eingabesequenz *Sequenz 6*.

# 6 Zusammenfassung

In dieser Arbeit wurde ein Algorithmus für *Motion Estimation* entwickelt und im Zuge eines Softwareprojekts ein Programm entwickelt, das diesen Algorithmus implementiert. Ziel war es, eine kurze Videosequenz zu analysieren und die einzelnen Bewegungsbereiche im Video durch affine Ebenenbewegungen abzubilden. Die Videosequenzen für den Test stammen zum einen Teil aus Motion-Standardsequenzen und zum anderen Teil aus historischem Videomaterial aus den 1920er und 1930er Jahren.

Im Gegensatz zu Stereo Vision werden Verdeckungen in Motion Estimation bisher noch kaum berücksichtigt. Darum wurde in dieser Arbeit ein Ansatz gewählt, welcher den Arbeiten von Jiangjian Xiao und Mubarak Shah [35, 36] stark nachempfunden ist. Ihre Arbeiten berücksichtigen Verdeckungen ausdrücklich und erzielen damit sehr gute Ergebnisse.

Dem Algorithmus liegt folgende Idee zugrunde: Anstatt nur zwei aufeinanderfolgende Bilder zu analysieren, sollen mehrere Bilder der Videosequenz betrachtet werden. Die Annahme dabei ist, dass durch die größere Bilderanzahl auch mehr Informationen zur Berechnung der Ebenen und ihrer affinen Bewegungen zur Verfügung stehen und damit exaktere Ergebnisse erzielt werden können. Eine weitere Annahme ist, dass bei einer kleinen Anzahl von Bildern die Bewegungsänderungen zwischen den Bildpaaren stetig ist. Innerhalb dieser Gruppe von Bildern dient das erste Bild als Referenzbild. Zu diesem Referenzbild werden die anderen Bilder der Gruppe in Beziehung gesetzt. Von der Energiefunktion wird die Bedingung gefordert, dass einem Bildpunkt des Referenzbilds über alle Bilder der Gruppe nur eine affine Bewegung oder der Verdeckungslabel zugeordnet werden kann. Die in der Gruppe der Bilder vorhandenen affinen Bewegungen werden durch einen vorausgehenden Region Growing und Region Merging Schritt erkannt. Dieser Algorithmusteil wird *Layer Extraction* Schritt bezeichnet. Der Algorithmusteil, der die Zuordnung zu den Bildpunkten macht, wird als *Layer Assignment* Schritt bezeichnet.

Der *Layer Extraction* Schritt scheint bei allen Videosequenzen sehr gute Beschreibungen für die affinen Bewegungen zu liefern. Der *Layer Assignment* Schritt wurde in dieser Arbeit angepasst und lieferte nicht so gute Ergebnisse, wie sie in den Arbeiten [35, 36] gezeigt werden.

Der Region Growing Schritt ist ein guter Ansatz, um eine affine Bewegung exakt zu bestimmen. Verbesserungen hinsichtlich der Robustheit sind nicht ausgeschlossen und wünschenswert. Ein Problem beim Region Growing Schritt ist, dass die Region in einen fremden Bereich hinauswächst und dadurch die affinen Parameter verfälscht werden. Eine Möglichkeit dieses Verhalten zu verhindern wäre, den Region Growing Schritt rechtzeitig abzubrechen. Denn zum einen ändern sich die affinen Parameter nach einigen Wachstumsphasen kaum mehr, zum anderen verschmilzt der Region Merging Schritt Regionen mit sehr ähnlichen affinen Parametern und berechnet aus den verschmolzenen Regionen

die affinen Parameter neu. Wenn zum Beispiel der Region Growing Schritt im Kalender der *Mobile & Calendar* Sequenz 5 Regionen mit leicht unterschiedlichen affinen Parametern bestimmt hat, dann kann der Region Merging Schritt diese 5 Regionen verschmelzen und aus einer großen Region die affinen Parameter des Kalenders neu berechnen. Es müsste allerdings ein vernünftiges Abbruchkriterium des Region Growing Schritts gefunden werden, das für alle möglichen Eingabesequenzen geeignet ist. Derzeit wird der Region Growing Schritt beendet, wenn die Region in einem Wachstumsschritt nicht größer wurde. Man könnte dieses Kriterium um eine maximale Iterationsanzahl erweitern. Oder man bestimmt, dass sich die affinen Parameter in jedem Wachstumsschritt mindestens um einen bestimmten Wert ändern müssen. Tun sie das nicht, wird der Region Growing Schritt abgebrochen. Wenn der Region Growing Schritt vorzeitig abgebrochen wird, wäre das außerdem ein signifikanter Gewinn in der Laufzeit des Programms.

Ein weiteres Potenzial um die Ergebnisse im Assignment Schritt zu verbessern wäre es, die Interpolation der affinen Parameter zwischen dem ersten und letzten Bild der Eingabesequenz zu verfeinern. Derzeit wird eine lineare Veränderung zugrunde gelegt. So ist die affine Bewegung, die im Bildpaar $\{1, 2\}$ auftritt, ein Viertel der Bewegung, die im Bildpaar $\{1, 5\}$ auftritt. Eventuell ergibt sich ein Vorteil daraus, wenn man den Pfad, den der Featurepoint zwischen dem ersten und letzten Bild beschreibt, für die Interpolation verwenden würde. Man könnte zum Beispiel die Abstandsverhältnisse bestimmen, die sich auf direktem Weg von den Koordinaten des Featurepoint im ersten Bild bis zu den Koordinaten im letzten Bild ergeben. Diese Abstandsverhältnisse könnten dann für die Interpolation verwendet werden. Durch diese Modifikation wäre nicht nur eine stetige Bewegung, sondern auch eine beschleunigte Bewegung gut im Layer Assignment Schritt berücksichtigt.

Bis jetzt wurden alle Parameter manuell eingestellt. Es ist sehr wahrscheinlich, dass sie nicht optimal bestimmt wurden. Ein Ground Truth wäre gut geeignet um das Ergebnis objektiv zu beurteilen. Das ließe sich dann auch automatisieren, sodass sich das Programm anhand des Ground Truths selbst parametrisieren könnte, bis es ein optimales Ergebnis findet. Dann ließe sich – anhand der optimalen Ergebnisse zu verschiedenen Eingabesequenzen – eventuell eine Regel aufstellen, welche aus den Eigenschaften einer Eingabesequenz die optimalen Parameter ableitet.

Die Bilder der Eingabesequenz werden nicht gleich behandelt, das Referenzbild hat eine besondere Stellung in der Energiefunktion. Eventuell ließe sich die Energiefunktion umschreiben, sodass alle Eingabebilder gleich wichtig sind und die Bildpaare symmetrisch behandelt werden. Mit symmetrisch ist gemeint, dass die Bewegung, die von Bild $A$ nach Bild $B$ stattfindet, umgekehrt auch von $B$ nach $A$ stattfinden muss. Ein wünschenswerter Nebeneffekt wäre, dass man in einem Layer Assignment Ablauf nicht nur die Bewegungen im Referenzbild bestimmt hat, sondern gleichzeitig auch die Bewegungen in allen anderen Bildern der Eingabesequenz. Eine Energiefunktion, in der die Eingabebilder symmetrisch behandelt werden, wird von Kolmogorov und Zabih in ihrer Arbeit [20] verwendet.

# Literaturverzeichnis


[1] S. Ayer, H. S. Sawhney: *Layered representation of motion video using robust maximum-likelihood estimation of mixture models and mdl encoding*, International Conference on Computer Vision, pp. 777, 1995.

[2] S. Birchfield, T. Thormaehlen: *An implementation of the Kanade-Lucas-Tomasi feature tracker*, http://www.ces.clemson.edu/ stb/klt/, Version 1.3.2, 23. Okt. 2006.

[3] M. Bleyer: *Segmentation-based stereo and motion with occlusions*, Dissertation an der Technischen Universität Wien, 2006.

[4] M. Bleyer, M. Gelautz: *A layered stereo matching algorithm using image segmentation and global visibility constraints*, ISPRS Journal of Photogrammetry & Remote Sensing 59, pp. 128-150, 2005.

[5] G. Borgefors: *Distance transformations in digital images*, Computer Vision, Graphics and Image Processing, vol. 34, pp. 344-371, 1986.

[6] Y. Boykov, V. Kolmogorov: *An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision*, IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 26, no. 9, pp. 1124-1137, 2004.

[7] Y. Boykov, V. Kolmogorov: *Maxflow - software for computing mincut/maxflow in a graph*, http://www.cs.ucl.ac.uk/staff/V.Kolmogorov/software.html, Version 3.0, 07. Aug. 2009.

[8] Y. Boykov, O. Veksler, R. Zabih: *A new algorithm for energy minimization with discontinuities*, Proceedings of the Second International Workshop on Energy Minimization Methods in Computer Vision and Pattern Recognition, pp. 205-220, 1999.

[9] Y. Boykov, O. Veksler, R. Zabih: *Energy minimization with discontinuities*, IEEE International Workshop on Energy Minimization Methods in Computer Vision, pp. 205-220, 1999.

[10] Y. Boykov, O. Veksler, R. Zabih: *Fast approximate energy minimization via graph cuts*, IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 23, no. 11, pp. 1222-1239, 2001.

[11] Th. H. Cormen, Ch. E. Leiserson, R. Rivest, C. Stein:, *Algorithmen - Eine Einführung*, 2. Auflage, Oldenbourg Wissenschaftsverlag GmbH, 2007.

[12] D. Greig, B. Porteous, A. Seheult: *Exact maximum a posteriori estimation for binary images*, Journal of the Royal Statistical Societ, vol. 51, no. 2, pp. 271-279, 1989.

[13] B. Horn, B. Schunck: *Determining optical flow*, Artificial Intelligence, vol. 17, pp. 185-203, 1981.

[14] Intel: *Open source computer vision library OpenCV*, http://sourceforge.net/projects/opencvlibrary/, Version 1.0, 18. Okt. 2006.

[15] T. Kanade, C. Tomasi: *Detection and tracking of point features*, Carnegie Mellon University Technical Report CMU-CS-91-132, April 1991.

[16] S. B. Kang, R. Szeliski, J. Chai: *Handling occlusions in dense multi-view stereo*, Conference on Computer Vision and Pattern Recognition, vol. 1, pp. 103-110, 2001.

[17] Q. Ke, T. Kanade: *A robust subspace approach to layer extraction*, Proceedings of the Workshop on Motion and Video Computing, pp. 37, 2002.

[18] Q. Ke, T. Kanade: *A subspace approach to layer extraction*, Conference on Computer Vision and Pattern Recognition, vol. 1, pp. 255, 2001.

[19] V. Kolmogorov, C. Rother: *Comparison of energy minimization algorithms for highly connected graphs*, European Conference on Computer Vision, pp. 1-15, 2006.

[20] V. Kolmogorov, R. Zabih: *Computing visual correspondence with occlusion via graph cuts*, International Conference On Computer Vision, pp. 508-515, 2001.

[21] V. Kolmogorov, R. Zabih: *Multi-camera scene reconstruction via graph cuts*, European Conference on Computer Vision, vol. 3, pp. 82-96, 2002.

[22] V. Kolmogorov, R. Zabih: *What energy functions can be minimized via graph cuts*, IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 26, no. 2, 2004.

[23] B. Lucas, T. Kanade: *An iterative image registration technique with an application to stereo vision*, Proceedings of Imaging Understanding Workshop, pp. 121-130, 1981.

[24] I. Patras, E. Hendirks, R. Lagendijk: *Video segmentation by map labeling of watershed segments*, IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 23, no. 3, pp. 326-332, 2001.

[25] D. Scharstein, R. Szeliski: *A taxonomy and evaluation of dense two-frame stereo correspondence algorithms*, International Journal of Computer Vision, pp. 7-42, 2002.

[26] J. Shi, C. Tomasi: *Good features to track*, Technical Report TR-93-1399, Cornell University, 1993.

[27] J. Shi, C. Tomasi: *Good features to track*, Conference on Computer Vision and Pattern Recognition, pp. 593-600, 1994.

[28] R. Szeliski, R. Zabih, D. Scharstein, O. Veksler, V. Kolmogorov, A. Agarwala, M. Tappen, C. Rother: *A comparative study of energy minimization methods for markov random fields*, European Conference on Computer Vision, vol. 2, pp. 19-26, 2006.

[29] R. Szeliski, R. Zabih: *An experimental comparison of stereo algorithms*, IEEE Workshop on Vision Algorithms, pp. 1-19, 1999.

[30] H. Tao, H. Sawhney, R. Kumar: *Object tracking with bayesian estimation of dynamic layer representations*, Transactions on Pattern Analysis and Machine Intelligence, vol. 24, no. 1, pp. 75-89, 2002

[31] O. Veksler: *Efficient graph-based energy minimization methods in computer vision*, A Dissertation Presented to the Faculty of the Graduate School of Cornell University, 1999.

[32] J. Y. A. Wang, E. H. Adelson: *Representing moving images with layers*, IEEE Transactions on Image Processing, vol. 3, issue 5, pp. 625-638, 1994.

[33] Wikipedia: *H.264*, http://de.wikipedia.org/wiki/H.264, 05. Feb. 2009

[34] J. Wills, S. Agarwal, S. Belongie: *What went where*, Conference on Computer Vision and Pattern Recognition, vol. 1, pp. 37-44, 2003.

[35] J. Xiao, M. Shah: *Motion layer extraction in the presence of occlusion using graph cut*, Conference on Computer Vision and Pattern Recognition, vol. 2, pp. 972-979, 2004.

[36] J. Xiao, M. Shah: *Accurate motion layer segmentation and matting*, Conference on Computer Vision and Pattern Recognition, vol. 2, pp. 698-703, 2005.

[37] Y. Zhou and H. Tao: *A background layer model for object tracking through occlusion*, International Conference on Computer Vision, vol. 2, pp. 1079-1085, 2003

# Abbildungsverzeichnis

# Lebenslauf

Christian Ammer, geboren am 11. Dezember 1975 in Kirchdorf an der Krems.

## Schulischer und wissenschaftlicher Werdegang

| | |
|---|---|
| 1982 - 1986 | Volksschule in Ried im Traunkreis |
| 1986 - 1990 | Hauptschule in Sattledt |
| 1990 - 1995 | HTL Elektrotechnik in Wels, Matura am 27. Juni 1995 |
| 1997 - 2003 | Studium der Wirtschaftsinformatik (175) an der Johannes Kepler Universität Linz |
| 2003 - 2005 | Bakkalaureatsstudium Wirtschaftsinformatik an der Universität Wien, Titel der Bakkalaureatsarbeit: "MobTel Datenbank mit WebService-Schnittstelle und graphischer Oberfläche" |
| 2005 - 2009 | Magisterstudium Wirtschaftsinformatik an der Universität Wien |

## Beruflicher Werdegang

| | |
|---|---|
| 1995 - 1996 | Bundesheer |
| 1996 - 1997 | Leasingarbeiter bei der Firma Project Consult in Linz |
| 1999 - 2000 | Programmierer bei der Firma CAS in Gmunden |
| 2000 - 2002 | Programmierer bei der Firma TAB in Ansfelden |
| seit 2003 | Softwareentwickler bei der Firma ADES in Wien |