Lehrstuhl für Multimediakommunikation und Signalverarbeitung
(LMS)
Universität Erlangen-Nürnberg
Prof. Dr.-Ing. A. Kaup

Bewegungskompensation für objektive Videoqualitätsmaße

Michael Balda

Bachelorarbeit

Hochschullehrer: Prof. Dr.-Ing. André Kaup
Betreuer: Marcus Barkowsky
22. Oktober 2004

Inhaltsverzeichnis

Liste der verwendeten Bezeichner

Bezeichner	Erklärung
$\mathbf{I_k}(\mathbf{x}), k \in \{L, a, b\}$	Farbkomponente eines Einzelbildes an der Position x
r_m^t	Region m zum Zeitpunkt t
g_m	Menge der Deskriptoren benachbarter Regionen
M_m^t	Größe der Region m zum Zeitpunkt t
$bb_{m,x}^t, bb_{m,y}^t$	Breite bzw. Höhe des umrandenden Rechtecks der Region m
h_m	3x3-Filtermaske eines gleitenden Mittelwertfilters
$r_{max}, r_{u,max}, r_{n,max}$	Maximalzahl gültiger / umstrittener / neuer Regionen
R_I	Segmentierungsmaske mit umstrittenen Regionen
R_E	Segmentierungsmaske mit verfolgten Regionen
R_N	Segmentierungsmaske mit neu detektierten Regionen
R	Segmentierungsmaske aus der Verschmelzung von R_E und R_N
$\text{hist}_{q,k}^{t-1}$, $\text{hist}_{q,k}^{t-1}$	(Normalisiertes) Histogramm der Region r_k für $q \in \{L, a, b\}$
P_k^t	Charakteristischer Wert einer Region r_k^t
$\text{sad}_{i,j}$	Summe der abs. Differenzen der Farbwerte in den Blocken i und j
$\mathcal{B}_i$	Menge der Pixel im Block i
$\mathbf{d}_j$	Durch Suchstratiegie vorgegebener Verschiebungsvektor
$B_{t,i}(u, v)$	Fourier-Transformation des Blocks $b_i(x, y)$
$P_{n,i}$	Produkt aus $B_{t-1,i}B_{t,i}^*$
$p_i(x, y)$	Verschiebungsoberfläche zum Block i
$F_i(x, y)$	Cosinus-Fensterfunktion
$\mathbf{v_x}$	Bewegungsvektor im Punkt $\mathbf{x}$
$v_{i,\phi}, \mathbf{v}_{i,r}$	Winkel und Betrag des Bewegungsvektors i
h_r, h_ϕ	Diskretisierungsstufenhöhe für Betrag und Winkel
$e_{i,r}, e_{i,\phi}$	Quantisierungslevel für Betrag und Winkel des Vektors i
$\mathbf{z}_j = (z_{j,r}, z_{j,\phi})$	Schwerpunkt für Klassse j
$\text{Dist}_{kmcc}, \text{Dist}_{km}, \text{Dist}_{maximin}$	Distanzmaß für KMCC-/K-Means-/Maximin-Algorithmus
$\mathbf{u}_k = (u_k, v_k)^T$	Bewegungsvektoren in kartesischen Koordinaten
u_{max}, v_{max}	Maximalwerte der Bewegungsvektoren
I_{max}	Abschätzung des maximalen Farbabstands der Pixel eines Bildes
$\overline{S}(r_k^t), \overline{I}(r_k^t), \overline{V}(r_k^t)$	Orts- / Farb- / Bewegungsschwerpunkt einer Region
S_{TH}, C_{TH}	Schwellwerte für Bewegungs- und Farbähnlichkeit
$d_F(r_k^0, r_l^0)$	Farbliche Differenz zwischen Regionenschwerpunkten
$D_U(r_m, r_n)$	Bewegungsähnlichkeitsdistanz zwischen zwei Regionen
$D_{U,k}$	miminale Bewegungsähnlichkeitsdistanz für k übrige Regionen
$\Upsilon_t(r_n^t)$	Präsenzindikator für Region zum Zeitpunkt t
$E_m^t(r_i)$	mittlerer Schätzfehler für Region i mit Bewegungsmodell für Region m

Kapitel 1

Einleitung

1.1 Motivation einer Bewegungsschätzung für objektive Videoqualitätsmessung

Im Rahmen dieser Arbeit soll eine Methode vorgestellt werden, der Bewegungsinformation auf Objektbasis für Videosequenzen ermittelt. Die Daten einer gewöhnlichen Bewegungsschätzung sollen dabei derart aufbereitet werden , dass sie einer objektiven Videoqualitätsmessung von Nutzen sein können.

Es soll ein Algorithmus vorgestellt, angepasst und implementiert werden, der in der Lage ist, für eine beliebige Videosequenz eine Objektsegmentierung durchzuführen und auf Basis dieser Segmentierung die Objektverfolgung zu betreiben. Daraus lassen sich Informationen über die Bewegungsrichtung eines oder mehrerer Objekte gewinnen, die Rückschlüsse darauf erlauben sollen, welche Bildbereiche im Fokus eines Betrachters des Videos liegen. Die Darstellungsqualität solcher Bereiche ist anschließend von einer objektiven Videoqualitätsmesssoftware höher zu bewerten, da Fehler oder Artefakte z.B. eines Coders hier besonders auffallen und somit den subjektiven Qualitätseindruck in wesentlich stärkerem Maße beeinflussen, als etwa ein gleichartiger Fehler in einem Bildbereich, der nicht die Aufmerksamkeit des Betrachters auf sich zieht. Eine denkbare Möglichkeit, diese Information sinnvoll in ein Videoqualitätsmessprogramm einfließen zu lassen, wäre zum Beispiel eine entsprechende Gewichtung der Pro-Pixel-Qualitätsindikatoren der Messsoftware mit den Gewichten aus der Bewegungskompensation.

1.2 Grundsätzliche Überlegungen

Damit es möglich ist, die Bewegungsrichtung von Objekten zu schätzen, ist es zunächst unbedingt erforderlich herauszufinden, welche Merkmale einen Bildbereich ausmachen, der etwas darstellt, was ein Betrachter als einheitliches Objekt bezeichnen würde.

Eine der wichtigsten dieser Eigenschaften ist eine einheitliche Bewegungsrichtung aller Elemente dieses Bildbereichs, deshalb ist es unverzichtbar, für alle weiteren Prozesse im Rahmen der Objektsegmentierung und -verfolgung auf möglichst verlässliche Bewegungsvektoren für die Bildelemente (meistens Blöcke aus mehreren Bildpunkten) zurückgreifen zu können. Deshalb wird im folgenden Kapitel kurz auf die verwendeten Methoden, deren Eigenschaften und Probleme eingegangen.

Eine weitere Eigenschaft, die ein semantisches Objekt ausmacht, ist die farbliche Homogenität, d.h. es wird davon ausgegangen, dass die Bildbereiche, welche zu einem Objekt gehören, farblich gewisse Ähnlichkeiten aufweisen. Desweiteren sollten diese Bildbereiche natürlich zusammenhängend sein und nicht etwa an verschiedenen Stellen in einem Einzelbild getrennt vorliegen.

Es sind durchaus noch weitere Merkmale denkbar, auf die man zur Objektsegmentierung zurückgreifen könnte, so können beispielsweise Informationen über die Textur eines Objekts oder Kanteninformationen bzw. Kantenbewegung einfließen. Diese kommen jedoch im Rahmen dieser Arbeit nicht zum Einsatz, weshalb hier nicht näher darauf eingegangen werden soll.

Da die Bewegungsschätzung eine wichtige Basis der Verfahren in dieser Arbeit ist, sollen zunächst stellvertretend zwei einfache Bewegungsschätzer vorgestellt werden, die zur Erzeugung der Bewegungsdaten zum Einsatz kamen. Das darauffolgende Kapitel beschäftigt sich mit einfachen Ansätzen zur Klassifizierung der Hauptbewegungsrichtungen in Einzelbildern einer Videosequenz. Anschließend wird ein komplexerer Algorithmus zur Objektsegmentierung und -verfolgung detailiert vorgestellt und auf dessen Implementierung eingegangen.

Der Algorithmus wurde in der Programmiersprache C implementiert und in die Videobearbeitungssoftware *Iconvert* integriert.

Kapitel 2

Verwendete Methoden zur Bewegungsschätzung

2.1 Grundsätzliches zur Bewegungsschätzung

Es existieren zahlreiche verschiedene Algorithmen zur Bewegungsschätzung in Videoseqeuenzen, die teilweise unterschiedlichen Anforderungen gerecht werden sollen. Die im Rahmen dieser Arbeit verwendeten Algorithmen sind die blockbasierte Bewegungsschätzung und die Phasenkorrelation. Beiden ist gemeinsam, dass sie die Bewegungsschätzung durch den Vergleich der Luminanzwerte eines Bildes einer Sequenz mit den Luminanzwerten des vorhergehenden Bildes vollziehen. Diese Bilder werden in (quadratische) Blöcke von 8x8 oder 16x16 Pixeln aufgeteilt und anschließend die wahrscheinliche Bewegung jedes Blocks von einem Bild zum nächsten geschätzt. Als Ergebnis erhält man einen zweidimensionalen Bewegungsvektor für jeden Block, der die Bewegung in horizontaler und vertikaler Richtung angibt. Die Bewegungsschätzung der Blöcke läuft bei beiden Algorithmen sehr unterschiedlich ab und hat somit individuelle Eigenschaften und Schwächen, auf die im folgenden kurz eingegangen werden soll.

2.2 Block-Matching

2.2.1 Vorgehensweise

Die blockbasierte Bewegungsschätzung gehört zu den einfachsten und zugleich populärsten Techniken zur Bewegungsschätzung. Der Algorithmus umfasst folgende Schritte:

Algorithmus

1. Teile das Bild gleichmäßig in Blöcke fester Größe
2. Für jeden Block i
   (a) Suche im vorangegangenen Bild unter Verwendung einer speziellen *Suchstrategie* den Block j gleicher Größe, der

die kleinste *Distanz* bezüglich der Luminanz der Blöcke zu i hat

(b) Berechne die Differenz zwischen den Mittelpunkten von i und j

(c) Weise diese Differenz dem Block i als Bewegungsvektor zu

2.2.1.1 Anmerkungen

Um die minimale *Distanz* in der Luminanz zwischen zwei Blöcken zu ermitteln verwendet man die Summe der quadratischen Differenzen oder die schneller zu berechnende Summe der absoluten Differenzen (SAD):

$$sad_{i,j} = \sum_{k \in \mathcal{B}_i} \mid I_t(\mathbf{k}) - I_{t-1}(\mathbf{k} - \mathbf{d_j}) \mid$$

wobei i der untersuchte Block im aktuellen Bild ist, j der Kandidat mit um $\mathbf{d_j}$ verschobener Position, $I_t(\mathbf{x})$ das Luminanz-Signal zum Zeitpunkt t an der Position $\mathbf{x}$, $\mathcal{B}_i$ die Menge der Positionen aller Pixel im Block i.

Die *Suchstrategie* gibt an, an welchen Positionen der Block aus dem vorhergehenden Bild gesucht werden soll, um die SAD zu minimieren. Ein einfacher Ansatz ist die sogenannte *Full Search*. Hier wird die SAD aller Blöcke innerhalb einer gegebenen Maximaldistanz untersucht und die Verschiebung $\mathbf{d_j}$ als Bewegungsvektor angenommen, die zum Block mit der minimalen SAD führt. Optimierte Strategien stützen sich z.B. auf die Annahme, dass die SAD mit zunehmender Distanz zum Minimum immer zunimmt.

Da bei der Untersuchung von Blöcken am Bildrand normalerweise Blöcke zum Vergleich herangezogen würden, die außerhalb des Bildes liegen würden, müssen für diesen Algorithmus die Ränder des vorhergehenden Bildes nach außen um die maximale Suchweite verlängert werden, z.B. dadurch dass die oberste und unterste Zeile bzw. die äußersten Spalten an den Rändern wiederholt werden.

2.2.2 Eigenschaften, Vor- und Nachteile

Das Block-Matching eignet sich besonders gut für den Einsatz der Bewegungs-Prädiktion in Videocodern, da es mit geeigneter Suchstrategie verhältnismäßig wenig Rechenaufwand erfordert und durch die Minimierung der Summe der absoluten Differenzen gewährleistet, dass die Differernz zwischen prädizierten Bild und tatsächlichem Bild sehr klein ist. Ein Nachteil besteht allerdings darin, dass in Bildbereichen mit besonders homogener Luminanz die absoluten Differenzen sehr ähnlich sind und der Algorithmus an solchen Stellen falsche stark vor der tatsächlichen Bewegungsrichtung abweichende Bewegungsvektoren liefern kann. Bei besonders schnellen Bewegungungen liegt der am Besten passende Block unter Umständen außerhalb der maximalen Suchweite, was ebenfalls zu unerwünschten Ergebnissen führt.

2.3 Phasenkorrelation

2.3.1 Vorgehensweise

Die Phasenkorrelation arbeitet ebenfalls auf Blockbasis, vergleicht aber nicht verschobene Blöcke, sondern versucht die Verschiebung innerhalb eines Blocks an fester Position in zwei aufeinanderfolgenden Bildern (genauer: deren Luminanzwerten) $b_{t,i}, b_{t-1,i}$ über deren Korrelation zu ermitteln. Zur Berechnung dieser Korrelation wird der Weg über die Fourier-Transfomation gewählt, da im Fourier-Raum die Korrelation zweier Blöcke der Multiplikation eines transformierten Blocks mit den konjugiert-komplexen Werten des anderen Blocks entspricht:

$$\mathcal{F}\{\sum_{\mathbf{n}} b_{t,i}(\mathbf{n}) b_{t-1,i}(\mathbf{n}-\mathbf{d})\} = B_{t,i}(\mathbf{u}) B^*_{t-1,i}(\mathbf{u})$$

Das Resultat geteilt durch den Betrag ergibt einen Phasenversatz, dieser kann dann in den Ortsbereich zurück transformiert werden und würde dort idealerweise einen Dirac-Stoß an der Position liefern, die der Verschiebung entspricht. In der Realität jedoch, da die Luminanz gewöhnlich keine mit der Blockgröße periodische Funktion darstellt, erhält man $d_x \times d_y$ diskrete Werte auf dem untersuchten Block, die eine Verschiebungsoberfläche darstellen (siehe Abb. 2.1). Deren Werte an den verschiedenen Positionen geben Aufschluss darüber, wie wahrscheinlich eine Verschiebung an die zu dem jeweiligen Wert gehörende Position ist. Die Position mit dem Maximalwert wird als Verschiebungsvektor gewählt.

2.3.1.1 Algorithmus

1. Zerlege Luminanzwerte des aktuellen Bildes und des vorhergehenden Bildes in Blöcke gleicher Größe $b_{t,i}(x,y)$, $b_{t-1,i}(x,y)$ mit

   $$b_{t,i} \equiv \bigcup_{x_i \in [i d_x;(i+1) d_x] \wedge y_i \in [i d_y;(i+1) d_y]} (x_i, y_i)$$

   (Blöcke optional überlappend (s. Eigenschaften), d.h. $\exists (i,j) : (i \neq j) \wedge (b_{t,i} \cap b_{t,j} \neq \emptyset)$)

2. Für jedes Paar $b_{t,i}, b_{t-1,i}$ aus entsprechenden Blöcken:

   (a) Optional: multipliziere Luminanzewerte $b_{t,i}(x,y)$ bzw. $b_{t-1,i}(x,y)$ der Bildpunkte (x,y) im Block mit *Fensterfunktion* $f_i(x,y)$

   (b) transformiere Blöcke $b_{t,i}$, $b_{t-1,i}$ mittels zweidimensionaler, diskreter Fourier-Transformation $B_{t,i}(u,v) = \mathcal{F}\{b_{t,i}(x,y)\}$, t-1 entsprechend

   (c) Für jedes entsprechende Wertepaar $B_{t,i}(u,v)$, $B_{t-1,i}(u,v)$ aus den Blöcken:

      i. berechne den konjugiert-komplexen Wert $B^*_{t,i}(u,v)$, (t-1 entsprechend)

      ii. multipliziere konjugiert-komplexe Wertepaare $P_i(u,v) = B_{t-1,i}(u,v) B^*_{t,i}(u,v)$

      iii. teile das Ergebnis durch den Betrag $P_{n,i}(u,v) = P_i(u,v) / \mid P_i(u,v) \mid$

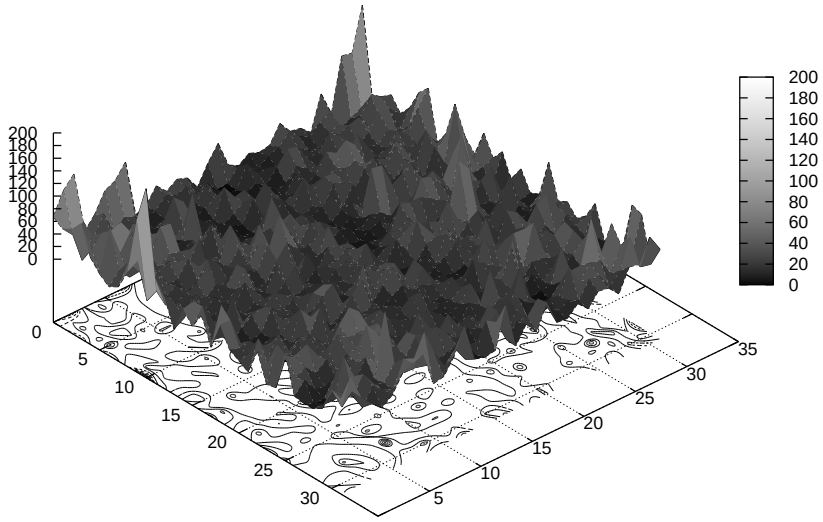

Abbildung 2.1: Oberflächenplot einer Verschiebungsoberfläche

(d) `berechne inverse, diskrete Fourier-Transformierte des Blocks` $p_i(x,y) = \mathcal{F}^{-1}\{P_{n,i}(u,v)\}$

(e) `berechne Absolutwerte` $p_{abs,i}(x,y) = \mid p_i(x,y) \mid$

(f) `finde Position des Maximalwerts im Block:` $(x_{max}, y_{max}) = \arg\max_{x,y}(p_{abs,i}(x,y))$

(g) `falls` $x_{max} > (d_x - 1)/2 : x_{max} = x_{max} - d_x$ `,` y_{max} `analog;` d_x `bzw.` d_y `sind die Dimensionen des Blocks in x- bzw. y-Richtung`

(h) `weise Block i den Bewegungsvektor` (x_{max}, y_{max}) `zu`

2.3.1.2 Anmerkungen

Es ist ratsam für die zweidimensionale Fourier-Transformation eine Implementation der schnellen Fourier-Transformation (FFT) zu verwenden, da damit eine erhebliche Geschwindigkeitssteigerung gegenüber der normalen diskreten Fourier-Transformation (DFT) erzielt werden kann.

Es besteht auch die Möglichkeit, diesen Algorithmus als Suchstrategie für das Blockmatching einzusetzen: Wähle statt dem Maximalwert die n größten Werte der Verschiebungsoberfläche und ermittle für diese Positionen die minimale SAD via Block-Matching.

2.3.2 Eigenschaften, Vor- und Nachteile

Die maximale Länge eines Bewegungsvektors enspricht der halben Blockdiagonale. Wird diese zu gering gewählt, ist bei schnelleren Bewegungen die Überschneidung in beiden Blöcken so gering, dass sich keine deutliche Spitze in der Verschiebungsoberfläche ausprägt und der Bewegungsvektor sehr unzuverlässig ist. Eine mögliche

Lösung dieses Problems besteht darin, dass man die n-fache Blockgröße (d_x bzw. d_y) wählt und als folgenden Block nicht den um d_x bzw. d_y verschobenen transformiert, sondern den um d_x/n bzw. d_y/n verschobenen. Man wählt also überlappende Blöcke und erhält die gleiche Auflösung der Bewegungsinformation und kann trotzdem eine größere Blockgröße wählen. Leider steigt bei Verwendung dieser Strategie auch der Rechenaufwand, z.B. doppelte Blockgröße in x- und y-Richtung mit Überlappung halber Blöcke, d.h. Verschiebung um $d_x/2$ bzw. $d_y/2$ statt d_x bzw. d_y würde zu vierfachem Rechenaufwand führen. Das Prädiktionsergebnis mit überlappenden Blöcken kann zusätzlich verbessert werden, wenn man eine Fensterfunktion auf die Blöcke anwendet, bevor sie transformiert werden, z.B. das *cos-Fenster:*

$$F_i(x,y) = \cos\frac{2\pi x}{d_x}\cos\frac{2\pi y}{d_y} I_t(b_{i,x}+x, b_{i,y}+y)$$

mit $b_{i,x}, b_{i,y}$ Mittelpunkt des Blocks i, $x \in [-d_x/2; d_x/2]$, y analog.

Kapitel 3

Einfache Ansätze der bewegungsbasierten Objektsegmentierung

In diesem Kapitel sollen zwei Ansätze vorgestellt werden, die ausschließlich auf Basis der Bewegungsvektoren alle Bereiche eines Einzelbildes in verschiedene Bewegungsklassen einteilen. Auf diese Weise sollen Aussagen darüber getroffen werden, welche Hauptbewegungsklassen vorliegen und welche Bildbereiche welchen Klassen zugeordnet werden können.

3.1 Quantisierung der Bewegungsvektoren

Bei der Quantisierung der Bewegungsvektoren werden die Vektoren zunächst von kartesischen Koordinaten (x,y) in Polarkoordinaten (Radius r, Winkel φ) umgerechnet, da eine Einteilung nach letzteren Kategorien sinnvoller erscheint. Anschließend werden alle Vektoren nach Winkel und Betrag (Länge) getrennt quantisiert und jeder Vektor wird einer Klasse zugeteilt. Diese Klassen werden nach der Anzahl der Mitglieder sortiert. Die 8 größten Klassen seien die Hauptbewegungsrichtungen, alle weiteren werden zu einer Auffangklasse zusammengefasst.

3.1.0.1 Algorithmus

1. `Transformiere jeden Bewegungunsvektor` $\mathbf{v}_i$ `von kartesischen in Polarkoordinaten` $\tilde{\mathbf{v}}_i$`:`
   - $\tilde{v}_{i,r} = \sqrt{v_{i,x}^2 + v_{i,y}^2}$
   - $\tilde{v}_{i,\varphi} = \arctan \frac{v_{i,y}}{v_{i,x}}$
2. `Ermittle maximalen Wert der Beträge aller Vektoren` r_{max}
3. `Setze Stufenhöhen für Betrag in Winkel in Abhängigkeit der Stufenzahl von Betrag (`s_r`) und Winkel (`s_φ`):`
   - `Betragsstufenhöhe:` $h_r = \frac{r_{max}}{s_r - 1}$
   - `Winkelstufenhöhe:` $h_\varphi = \frac{2\pi}{s_\varphi - 1}$

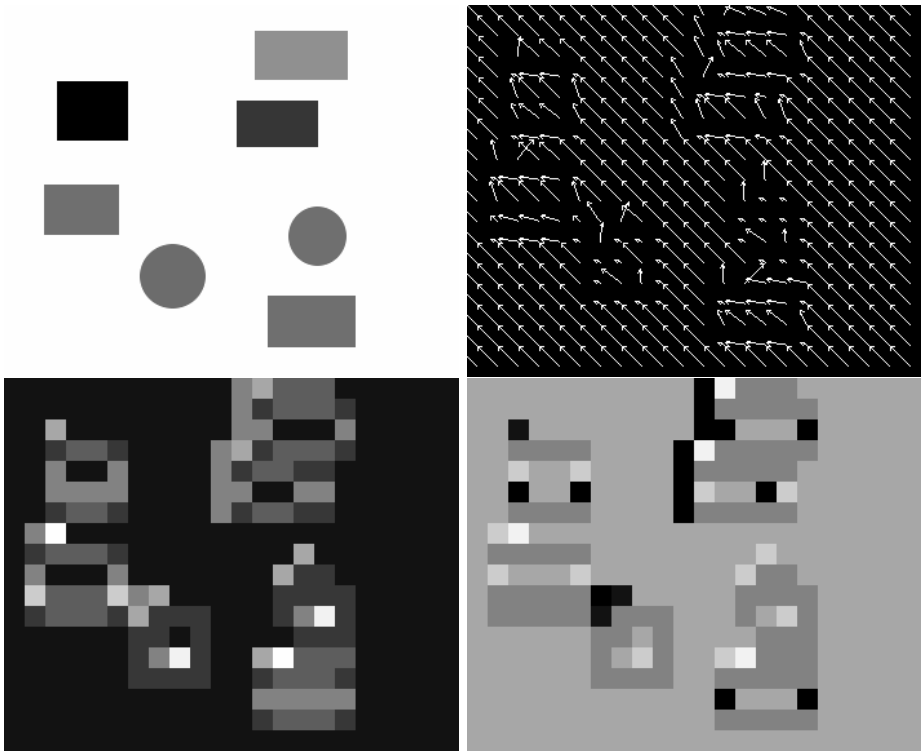

Abbildung 3.1: Zweites Bild einer synthetischen Beispielsequenz; Vektoren durch Block-Matching; Segmentierung mittels Quantisierter Bewegungsvekoren; mittels k-Means

4. Für alle Bewegungsvektoren $\tilde{\mathbf{v}}_i$:

   (a) Berechne Quantisierungslevel bezüglich Betrag und Winkel:

   - $e_{i,r} = \lfloor \frac{\bar{v}_{i,r} + \frac{h_r}{2}}{h_r} \rfloor$
   - $e_{i,\varphi} = \lfloor \frac{\bar{v}_{i,\varphi} + \frac{h_\varphi}{2}}{h_\varphi} + \frac{s_\varphi}{2} \rfloor$ (positiv [1])

   (b) Berechne Klassenummer für Vektor: $c_i = e_{i,r} s_\varphi + e_{i,\varphi}$

5. Sortiere alle Klassen absteigend nach der Anzahl der Elemente

6. Definiere Klassen neu:

   (a) Wähle die acht größten Klassen $c_j; j \in g$ (g ist Menge der Klassenindizes der acht größten Klassen) als neue Klassen $\tilde{c}_i, i \in [0; 7]$

   (b) Teile jedem Vektor, der in einer der größten Klassen c_j liegt die entsprechende neue Klasse $\tilde{c}$ zu

   (c) Weise verbleibende Vektoren der Auffangklasse $\tilde{c}_8$ zu

3.1.0.2 Anmerkungen

Ein potentielles Problem dieses Algorithmus wird hier nicht gelöst: Die Bewegungsvektoren mit einem Winkel etwas kleiner als π bzw. etwas größer als $-\pi$ werden von diesem Algorithmus als am weitesten entfernte Winkel aufgefasst. Es ist somit sehr unwahrscheinlich, dass Vektoren dieser Art der selben Klasse zugewiesen werden, selbst wenn sie gleich lang und somit sehr ähnlich sind.

3.2 Einteilung in Bewegungsklassen mittels k-Means-Clustering

Auch bei der Einteilung mittels k-Means werden wieder Bewegungsvektoren in Polarkoordination verwendet. Die Einteilung in Klassen erfolgt jedoch auf unterschiedliche Weise. Alle Vektoren werden den jeweils nächsten *Zentroiden* zugeordnet. Diese Zentroiden sind eine Art Schwerpunkte einer jeweiligen Klasse. Als Anfangspositionen der Zentroiden werden Punkte gewählt die möglichst gleichmäßig im *Merkmalraum*[2] verteilt sind. Anschließend wird für alle Klassen über die Merkmalvektoren der jeweiligen Klasse gemittelt und somit die neue Position der Zentroiden bestimmt. Dann werden wieder alle Merkmalvektoren einem Zentroiden mittels Nächster-Nachbar-Klassifikation zugewiesen und wieder gemittelt. Der Algorithmus terminiert, wenn alle Zentroiden ihre Positionen nur noch um weniger als einen Grenzwert d_g ändern. Eine Variante dieses Algorithmus kommt auch im Objektsegmetierungsalgorithmus vor, der später vorgestellt wird.

3.2.0.3 Algorithmus

1. Weise allen Zentroiden $\mathbf{z}_j$, $j \in [0, 7]$ Anfangswerte zu:

[1] $+\frac{s_\varphi}{2}$ ist erforderlich, da für den Winkel gilt $\varphi \in [-\pi; \pi]$, also sind negative Quantisierungsebenen möglich, die Klassennummern sollen aber positiv sein.

[2] Der Merkmalraum wird definiert durch die Komponenten der Eigenschaftsvektoren, hier also Betrag und Winkel der Bewegungsvektoren

- Betragskomponente: $z_{j,r} = \frac{1}{2} r_{max}$; r_{max} ist der maximale Betrag aller Bewegungsvektoren
- Winkelkomponente: $z_{j,\varphi} = \frac{j-4}{8} 2\pi$

2. Für alle Bewegungsvektoren $\tilde{\mathbf{v}}_i$: Füge $\tilde{v}_i$ der Menge g_k der zum nächsten Zentroiden $\mathbf{z}_k$ mit $k = \arg\min_j \{ \sqrt{(\frac{z_{j,r} - \tilde{v}_{i,r}}{r_{max}})^2 + (\frac{z_{j,\varphi} - \tilde{v}_{i,\varphi}}{2\pi})^2} \}$ gehörenden Vektoren zu

3. Berechne neuen Schwerpunkt für alle Klassen $k \in [0, 7]$

   (a) Speichere alte Position des Zentroiden: $\mathbf{z}_{k,alt} = \mathbf{z}_k$

   (b) Mittle über alle $\tilde{v}_i \in g_k$ und weise das Mittel $\mathbf{z}_k$ zu: $\mathbf{z_k} = \frac{\sum_{i \in g_k} \tilde{v}_i}{|g_k|}$

4. Wenn ein k existiert, für das gilt: $|\mathbf{z}_{k,alt} - \mathbf{z}_k| > d_g$, wiederhole ab Schritt 2

3.3 Bewertung der einfachen Ansätze

Eine Objektverfolgung, die sich lediglich auf die Bewegungsinformation stützt, führt leider in der Regel zu ungenügenden Ergebnissen (s. Abb. 3), deshalb erweisen sich diese Ansätze als nicht ausreichend, um tatsächlich Rückschlüsse auf den Fokus eines Betrachters zu ziehen. Diese Methoden können wesentliche Anforderungen nicht erfüllen: Zu einer Klasse gehörende Objekte weisen keine Konnektivität auf, d.h. unzusammenhängende Bildbereiche können derselben Klasse zugeordnet werden. Es wird keine Orts- oder Farbinformation miteinbezogen, was zur Folge hat, dass optisch vollkommen unterschiedliche Bereiche als zusammengehörig ausgewiesen werden. Durch die alleinige Fokussierung auf die Bewegungsvektoren gibt es kaum eine Möglichkeit, das Segmentierungsergebnis gegen Fehler durch falsch prädizierte Bewegungsvektoren abzusichern (Bilder erzeugen und kommentieren). Außerdem bergen diese Methoden den Nachteil in sich, dass sie nicht in der Lage sind, einmal segmentierte Bereiche in den folgenden Bildern mit der selben Klassennummer weiterzuverfolgen, d.h. ein und dasselbe semantische Objekt erhält in verschiedenen Einzelbildern normalerweise nicht immer den gleichen Klassenbezeichner, was aber erforderlich wäre, wollte man dem Fokus eines Betrachters über die Zeit folgen.

Kapitel 4

Objektsegmentierung und -verfolgung

Die Versuche mit den einfachen Ansätzen aus dem vorherigen Kapitel haben gezeigt, dass es erforderlich ist, einen Algorithmus zu verwenden, der auf mehr Informationen als nur die Bewegung zurückgreift und diese angemessen miteinenander verbindet. Damit soll ein Einzelbild auf eine Weise untergliedert werden, die in der Wahrnehmung eines Beobachters sinnvoll erscheint. Diese anfängliche Einteilung wird dann jeweils den Änderungen in den Folgebildern entsprechend angepasst [1].

4.1 Allgemeiner Aufbau

Die Objektsegmentierung gliedert sich grundsätzlich in zwei Phasen (vgl. Abb. 4.1): Die Intra-Frame-Segmentierung, die das erste Bild einer Sequenz bzw. das erste Bild nach einem Szenenwechsel in Regionen aufteilt und die anschließende Regionenverfolgung bzw. Detektion neuer Regionen. Beide greifen auf die Daten einer vorgeschalteten Bewegungsschätzung zurück, auf die bereits im zweiten Kapitel eingegangen wurde.

Der Algorithmus arbeitet grundsätzlich auf Bildern im *L*a*b*-Farbraum*, also muss jedes Bild einer Videosequenz zunächst in diesen Farbraum konvertiert werden. Der

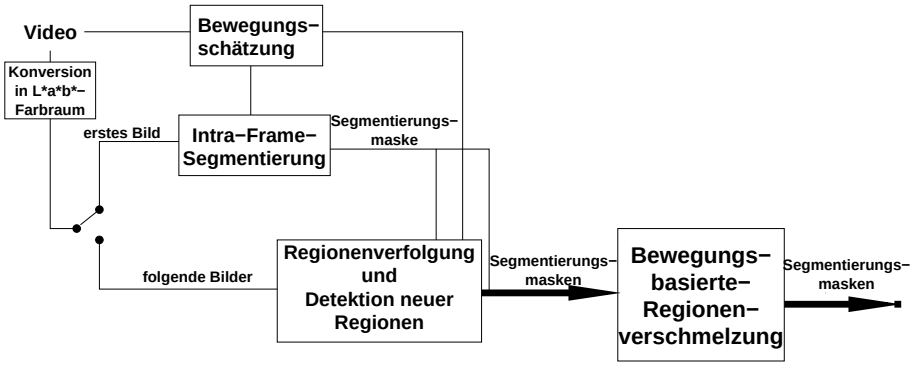

Abbildung 4.1: Allgemeiner Aufbau der Objektsegmentierung und -verfolgung

L*a*b*-Farbraum hat den entscheidenden Vorteil, dass die Abstände zwischen zwei Farben in etwa den farblichen Unterschied wiedergeben, der der menschlichen Wahrnehmung entspricht. So können zum Beispiel im L*a*b*-Raum zwei unterschiedliche Blautöne einen wesentlich geringeren Abstand aufweisen als zwei Grüntöne, obwohl diese im RGB-Farbraum den gleichen Abstand hätten. Die Ursache hierfür liegt darin, dass die menschliche Wahrnehmung Grüntöne besser auflößt als Blautöne. Die Umrechnung zwischen RGB und L*a*b* kann mittels dieser Formeln erfolgen [3]:

RGB in XYZ

$$\begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = \begin{pmatrix} S_r X_r & S_r Y_r & S_r Z_r \\ S_g X_g & S_g Y_g & S_g Z_g \\ S_b X_b & S_b Y_b & S_b Z_b \end{pmatrix} \begin{pmatrix} R^\gamma \\ G^\gamma \\ B^\gamma \end{pmatrix}$$

$$X_r = \frac{x_r}{y_r}; Y_r = 1; Z_r = \frac{1 - x_r - y_r}{y_r}$$

$$X_g = \frac{x_g}{y_g}; Y_g = 1; Z_g = \frac{1 - x_g - y_g}{y_g}$$

$$X_b = \frac{x_b}{y_b}; Y_b = 1; Z_b = \frac{1 - x_b - y_b}{y_b}$$

$$\begin{pmatrix} S_r \\ S_g \\ S_b \end{pmatrix} = \begin{pmatrix} X_r & Y_r & Z_r \\ X_g & Y_g & Z_g \\ X_b & Y_b & Z_b \end{pmatrix} \begin{pmatrix} X_w \\ X_w \\ Z_w \end{pmatrix}$$

mit $R, G, B \in [0; 1]$; γ Gammawert; X_w, Y_w, Z_w Referenzweiß; $(x_r, y_r), (x_g, y_g), (x_b, y_b)$ Chromatizitätskoordinaten des RGB-Farbraums. Beispielwerte:
CIE RGB: $\gamma = 2,2$, Referenzweiß: E, $(x_r, y_r) = (0.7350, 0.2650)$, $(x_g, y_g) = (0.2740, 0.7170)$, $(x_b, y_b) = (0.1670, 0.0090)$
PAL/SECAM RGB: $\gamma = 2,2$, Referenzweiß: D65, $(x_r, y_r) = (0.6400, 0.3300)$, $(x_g, y_g) = (0.2900, 0.6000)$, $(x_b, y_b) = (0.1500, 0.0600)$

XYZ in LAB

$$L = 116 f_y - 16; a = 500(f_x - f_y); b = 200(f_y - f_z)$$

mit

$$f_x = \begin{cases} \sqrt[3]{x_r} & x_r > \epsilon \\ \frac{\kappa x_w + 16}{116} & x_w \leq \epsilon \end{cases}; f_y = \begin{cases} \sqrt[3]{y_r} & y_r > \epsilon \\ \frac{\kappa y_w + 16}{116} & y_w \leq \epsilon \end{cases}; f_z = \begin{cases} \sqrt[3]{z_r} & z_r > \epsilon \\ \frac{\kappa z_w + 16}{116} & z_w \leq \epsilon \end{cases}$$

$$x_r = \frac{X}{X_w}; x_r = \frac{Y}{Y_w}; x_r = \frac{Z}{Z_w}$$

$\epsilon := 0,008856$; $\kappa := 903,3$ (CIE Standard)

4.2 Intra-Frame-Segmentierung

Die Intra-Frame-Segmentierung kommt immer dann zum Einsatz, wenn keine Segmentierungsmaske für das vorherige Einzelbild vorliegt oder diese nicht zum aktuellen Bild passt, also im ersten Einzelbild, nach Szenenwechseln oder Schnitten.

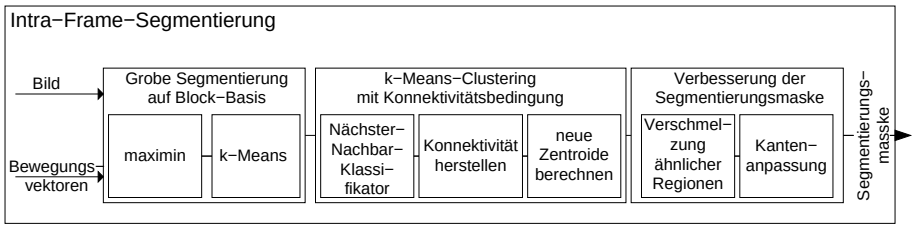

Abbildung 4.2: Allgemeiner Aufbau der Intra-Frame-Segmentierung

Sie selbst basiert nur auf einem Einzelbild, zu welchem aber Bewegungsvektoren vorliegen müssen, die durch den Vergleich mit dem n'chsten Bild in der Sequenz erzeugt werden müssen.

Die Intra-Frame-Segmentierung besteht aus drei Phasen (vgl. Abb. 4.2): Die grobe Segmentierung, mit der sinnvolle Startwerte für die folgenden Schritte errechnet werden, anschließend folgt das k-Means-Clustering mit Konnektivitätsbedingung, das die eigentliche Segmentierung erzeugt, die anschließend im Rahmen einer Segmentierungsmaskenverbesserung einigen Korrekturen unterzogen wird. Das Ergebnis ist ein zweidimendionales Feld $R_0(x, y)$, dessen Dimensionen genau dem des Eingabebildes entsprechen und das für jeden Bildpunkt einen ganzzahligen Wert enthält, der der Regionsnummer des entsprechenden Pixels entspricht. Die Regionsnummer liegt im Wertebereich von $[0; r_{max,0} - 1]$ wobei $r_{max,0}$ die Anzahl der gefundenen Regionen im ersten Bild ist. Aus dieser Maske kann nun zur Visualisierung des Segmentierungsergebnisses ein Graustufenbild $I_{A,0}(x, y)$ erzeugt werden, in dem jeder Region ein entsprechender Grauwert zugewiesen ist, z.B. $I_{A,0}(x, y) = R_0(x, y)\frac{255}{r_{max,0}}$ für 256 Graustufen.

Merkmalsvektoren und Distanzmaß Da die Intra-Frame-Segmentierung auf die Merkmale Bewegung, Farbe und Ort zurückgreift, ist es erforderlich ein *Distanzmaß* einzuführen, das es erlaubt unter Berücksichtigung dieser Eigenschaften eine Distanz zwischen einer Region r_k^t und einem Bildpunkt an Position $\mathbf{x}$ (evtl. auch einem Block b_i) zu berechnen. Hier kommt — soweit nicht anders angemerkt — grundsätzlich dieses Distanzmaß zum Einsatz:

$$\mathrm{Dist}_{kmcc}(\mathbf{x}, r_k^0) = \|I_0(\mathbf{x}) - \overline{I}(r_k^0)\| + \lambda_1\|\mathbf{V}_0(\mathbf{x}) - \overline{\mathbf{V}}(r_k^0)\| + \lambda_2\frac{\overline{A}}{A_k}\|\mathbf{x} - \overline{\mathbf{S}}(r_k^0)\|$$

mit $\overline{I}(r_k^0)$ Schwerpunkt der Region r_k^0 im L*a*b*-Farbraum; $\overline{V}(r_k^0)$ Mittel der Bewegungsvektoren; $\overline{\mathbf{S}}(r_k^0)$ Orts-Schwerpunkt und A_k Größe der Region k zum Zeitpunkt 0 (erstes Bild); $\overline{A}$ Mittlere Größe aller Regionen und

$$\lambda_1 = 2 \cdot \frac{I_{max}}{\sqrt{(2u_{max})^2 + (2v_{max})^2}}; \lambda_2 = 0,1 \cdot \frac{I_{max}}{\sqrt{p_{x,max}^2 + p_{y,max}^2}}$$

mit I_{max} Abschätzung des maximalen Farbabstandes; u_{max}, v_{max} maximale Bewegungsvektorlänge und $p_{x,max}, p_{y,max}$ Anzahl der Elemente (Pixel) in horizontaler und vertikaler Richtung.

Die Beträge der einzelnen Vektoren werden also mit deren möglichen Maximalwerten skaliert. Der Ausdruck $\frac{\overline{A}}{A_k}$ soll die Schaffung großer Regionen begünstigen, denn dieser Bruch liefert einen kleinen Wert für verhältnismäßig große Regionen und umgekehrt.

4.2.1 Grobe Segmentierung

Die grobe Segmentierung wird nicht auf Pixeln sondern auf Blöcken vollzogen, die die Eigenschaften mehrerer Bildpunkte zusammenfassen. Es müssen also zunächst Eigenschaftsvektoren für jeden Block ermittelt werden. Die Blöcke sind $d_x \times d_y$ Bildpunkte groß, hierfür werden zur Vereinfachung der Berechnung die gleichen Blockgrößen verwendet, die schon in der Bewegungsschätzung verwendet wurden. Die Bewegungsinformation $\mathbf{V}(x_b, y_b)$ ist also bereits in Blockkoordination vorhanden und bezeichnet die Bewegunginformation die aus dem Vergleich eines Bildes zum Zeitpunkt t mit dem vorherigen Bild zum Zeitpunkt t-1 gewonnen wurde. Die Ortsinformation eines Blocks liegt implizit vor, da die grobe Segmentierung auf Koordinaten $\mathbf{x}_b = (x_b, y_b)$ arbeitet, die einzelne Blöcke und nicht Pixel beschreiben, mit $b(x_b, y_b)$ als die Menge aller Bildpunkte in diesem Block. So verweist $(x_b, y_b) = (1, 3)$ also auf den Block mit der zweiten Position in horizontaletr und vierten in vertikaler Richtung. Die Farbwerte $\mathbf{I}_b(x_b, y_b)$ bzw. $\mathbf{I}(b_i)$ für jeden Block werden durch Mittelung der Werte aller Pixel in einem Block ermittelt:

$$\mathbf{I}_b(x_b, y_b) = \frac{\sum_{(x,y)\in b(x_b,y_b)} \mathbf{I}(x, y)}{d_x d_y}$$

4.2.1.1 Maximin-Algorithmus

Mittels dieser Daten auf Blockbasis wird zunächst eine anfängliche Klassenanzahl mit dem Maximin-Algorithmus ermittelt [5].

Der Maximin-Algorithmus stellt eine einfache Möglichkeit dar, beliebige Eigenschaftsvektoren zu Klassen zuzuweisen und gleichzeitig eine günstige Anzahl an Klassen zu erzeugen. Der Algorithmus wählt zunächst zufällig einen Block aus und definiert diesen Block als erstes Klassenzentrum $\mathbf{x}_{b,z1}$. Dann wird für alle anderen Blöcke die Distanz zu eben diesem Klassenzentrum ermittelt und dabei festgestellt, welcher Block die größte Distanz aufweist. Dieser wird zweites Klassenzentrum $\mathbf{x}_{b,z2}$. Die Distanz zwischen den ersten beiden Zentren wird *Skalierungsdistanz* genannt. Als Distanzmaß verwenden wir hierbei:

$$\mathrm{dist}_{maximin}(\mathbf{x}_{b,i}, \mathbf{x}_{b,j}) = |\mathbf{I}((\mathbf{x}_{b,i})) - \mathbf{I}(\mathbf{x}_{b,j})| + \frac{I_{max}}{\sqrt{x_{b,max}^2 + y_{b,max}^2}} |\mathbf{x}_{b,i} - \mathbf{x}_{b,j}|$$

$\mathbf{x}_{b,max} = (x_{b,max}, y_{b,max})$ sind die maximalen Blockkoordinaten in horizontaler bzw. vertikaler Richtung; I_{max} ist eine Abschätzung der größten Farbdistanz zweier Pixel im Bild.

Algorithmus

1. Wähle zufälligen Block b_{z1} an Position $\mathbf{x}_{b,z1}$ als erstes Klassenzentrum
2. Ermittle Block b_{z2} an Posititon $\mathbf{x}_{b,z2}$ = $\arg\max_{\mathbf{x_b}} \mathrm{dist}_{maximin}(\mathbf{x}_{b,z1}, \mathbf{x}_b)$ mit größter Distanz D_s = $\mathrm{dist}_{maximin}(\mathbf{x}_{b,z1}, \mathbf{x}_{b,z2})$ als zweites Klassenzentrum
3. Setze $z_{max} = 2$; z_{max} maximale Zahl an Klassenzentren
4. Für alle Blöcke b_i:
   Ermittle $D_{min}(i) = \min_{j\in[1;z_{max}]} = \mathrm{dist}_{maximin}(\mathbf{x}_{b,z_j}, \mathbf{x}_{b,i})$: Distanz zum nächsten Klassenzentrum

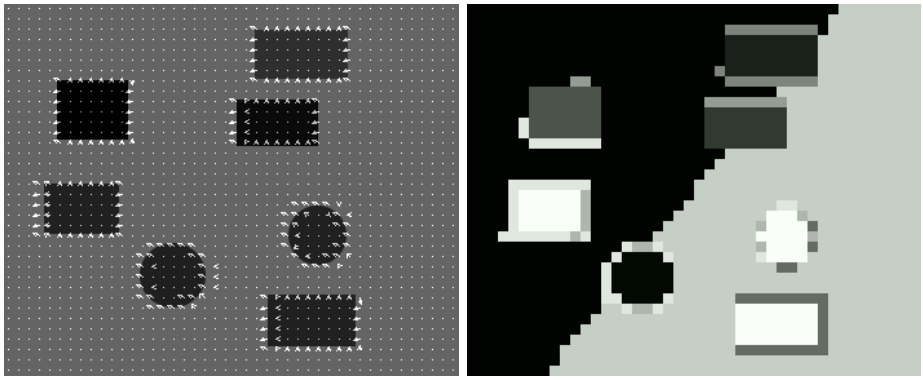

Abbildung 4.3: synthetische Beispielsequenz mit Bewegungsvektoren; Ergebnis der Maximin-Segmentierung

5. Ermittle $D_{max} = \max_{\forall i} D_{min}(i)$ größte Distanz zum nächsten Klassenzentrum
6. Falls $D_{max} > \frac{D_s}{3}$:
   (a) Setze $z_{max} = z_{max} + 1$
   (b) Neues Klassenzentrum $b_{z_{max}} = b_i$ mit $i = \arg\max_{\forall j} D_{min}(j)$
   (c) Wiederhole ab Schritt 4

Ergebnis Die Segmentierung nach Anwendung des Maximin-Algorithmus ist naturgemäß äußerst grob und noch relativ weit entfernt von der endgültigen Segmentierungsmaske. Deshalb werden die Ergebnisse der Vorsegmentierung hier lediglich für eine synthetische Sequenz vorgestellt, bei der der Zusammenhang zwischen Originalbild und der groben Segmentierung auf Blockbasis noch leicht zu erkennen ist. Abbildung 4.3 links zeigt ein Einzelbild einer synthetischen Beispielsequenz mit den dazugehörigen Bewegungsvektoren. Das rechte Bild zeigt die dazugehörige Segmentierungsmaske nach der Maximin-Segmentierung. Jede der Regionen wird dabei durch einen Grauwert repräsentiert. Der Hintergrund ist in zwei Regionen aufgeteilt und die Objektränder werden teilweise anderen Regionen zugeschlagen. Letzteres liegt vor allem an der Mittelung der Pixel-Farbwerte, wodurch ein Block am Objektrand eine eigene Farbe zugewiesen bekommt und somit als eigene Region erkannt werden kann. Die drei Objekte gleicher Farbe in der unteren Bildhälfte werden alle derselben Region zugeschlagen (in der Segmentierungsmaske weiß). Einen Teil dieser negativen Effekte soll die folgende K-Means-Segmentierung beheben, die die Position der Regionsschwerpunkte optimiert und somit etwa eine einheitliche Segmentierung des Hintergrundes ermöglicht. Speziell das Problem, dass unzusammenhängende Objekte der selben Region zugerechnet werden, kann aber erst durch die Einführung der *Konnektivitätsbedingung* nach der Vorsegmentierung behoben werden, da die Vorsegmentierung lediglich Startwerte liefern soll und eine Anwendung der Konnektivit'tsbedingung erst bei der Segmentierung auf Pixelbasis sinnvoll ist.

4.2.1.2 k-Means-Clustering

Mit der Anzahl z_{max} an Klassen und den Bewegungs-, Orts- und Farbwerten der Klassenzentren b_{z_i} als Anfangswerte für die Schwerpunkte wird nun ein K-Means-Clustering vollzogen: Zunächst werden alle Blöcke dem jeweils nächsten Zentroiden

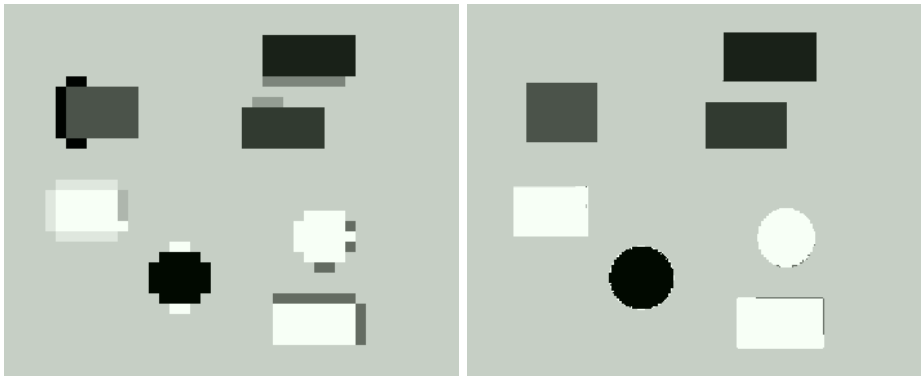

Abbildung 4.4: Ergebnis des K-Means-Algorithmus; Ergebnis der Vorsegmentierung auf Pixelkoordinaten

zugewiesen und anschließend wird für alle Zentroide über alle zu der jeweiligen Klasse gehörigen Blöcke gemittelt und somit neue Zentroide errechnet. Dies wird so lange wiederholt, bis die Zentroide ihre Position beibehalten. Als DistanmaßDist_{km} wird das bereits defininierte $\text{Dist}_{\text{kmcc}}$ verwendet, allerdings ohne den Faktor zur Gewichtung unterschiedlich großer Regionen $\frac{\overline{A}}{A_k}$, da das Konzept der Regionsgröße erst sinnvoll ist, sobald von im Bildbereich zusammenhängenden Regionen gesprochen werden kann (s. *Konnektivitätsbedingung*). Im folgenden wird also angenommen $\overline{A} = 1$ und $A_k = 1 \forall k$.

Algorithmus

1. Weise allen Zentroiden $\mathbf{z}_j$, $j \in [0, z_{max}]$ die Bewegungs- und Farbwerte sowie die Koordinaten $(\mathbf{I}, \mathbf{V}, \mathbf{x})$ der Blöcke b_{z_j} zu
2. Für alle Blöcke b_i: Füge b_i der Menge g_k der zum nächsten Zentroiden $\mathbf{z}_k$ mit $k = \arg\min_j \{\text{dist}_{km}(b_i, \mathbf{z}_j)\}$ gehörenden Blöcke zu
3. Berechne neuen Schwerpunkt für alle Klassen $k \in [0, z_{max}]$
   (a) Speichere alte Position des Zentroiden: $\mathbf{z}_{k,alt} = \mathbf{z}_k$
   (b) Mittle über alle $\mathbf{I}_i$, $\mathbf{V}_i$ und $\mathbf{x}_i$ der Blöcke $b_i \in g_k$ und weise das Ergebnis $\mathbf{z}_k$ zu:

$$\mathbf{z_k} = (\overline{\mathbf{I}}_k^0, \overline{\mathbf{V}}_k^0, \overline{\mathbf{S}}_k^0) = \left(\frac{\sum_{i \in g_k} \mathbf{I}_i}{|g_k|}, \frac{\sum_{i \in g_k} \mathbf{V}_i}{|g_k|}, \frac{\sum_{i \in g_k} \mathbf{x}_i}{|g_k|} \right)$$

4. Wenn ein k existiert, für das gilt: $|\mathbf{z}_{k,alt} - \mathbf{z}_k| > d_g$, wiederhole ab Schritt 2

Ergebnis In Abbildung 4.4 links kann man das Ergebnis der K-Means-Segmentierung auf Blockbasis sehen: Der Hintergrund wird jetzt nicht mehr in unterschiedliche Regionen aufgeteilt. Jedoch zeigen sich an den Objekträndern noch unerwünschte Effekte, die aber etwas abgemildert werden konnten: Die Fläche der Regionen an den Objekträndern hat deutlich abgenommen. Ein weiterer unerwünschter Effekt tritt bei den drei weißen Objekten im unteren Bildbereich auf:

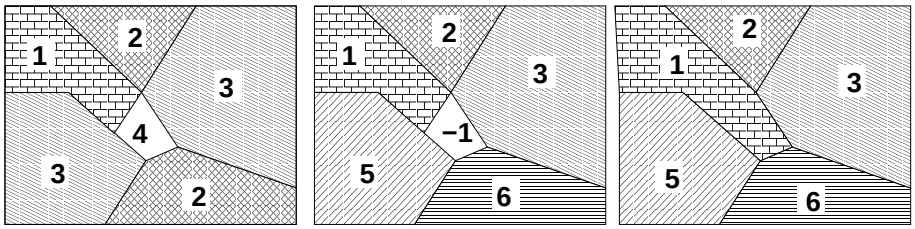

Abbildung 4.5: Links: Beispiel einer Segmentierung nach k-Means; Mitte: neue Regionen nach Herstellung der Konnektivität; rechts: nach neuer Zuweisung zu kleiner Regionen

Diese werden immer noch alle drei der gleichen Region zugerechnet, obwohl sie nicht miteinander verbunden sind und somit nicht einem einzigen Objekt zugeordnet werden können. Das rechte Bild zeigt das Ergebnis der Vorsegmentierung nach der Umrechnung auf Pixelkoordinaten. Die Blockkoordinaten der Zentroide der K-Means-Segmentierung wurden in Pixelkoordinaten umgerechnet und anschließend jeder einzelne Bildpunkt mittels Nächster-Nachbar-Klassifikation einem Zentroiden zugeordnet.

4.2.2 k-Means-Clustering mit Konnektivitätsbedingung

Mit dem k-Means-Clustering ist die Vor-Segmentierung beendet und die eigentliche Segmentierung mittels *k-Means mit Konnektivitätsbedingung* kann beginnen. Der Algorithmus arbeitet wie k-Means mit Zentroiden, denen die Bildelemente (jetzt auf Pixel-Basis) mittels Nächster-Nachbar-Klassifikation zugewiesen werden. Allerdings wird zur Berechnung neuer Positionen für die Zentroide ein anderes Verfahren angewandt.

Die Anzahl der Zentroide ist anfänglich die gleiche wie bei der Vor-Segmentierung mit k-Means. Als Startwerte werden die Werte der Zentroide aus der Vor-Segmentierung verwendet, allerdings werden die Ortsschwerpunkte auf Pixelkoordinaten umgerechnet: $\overline{\mathbf{S}}_k^0 = \overline{\mathbf{S}}_k^0 \cdot (d_x, d_y) + (\frac{d_x}{2}, \frac{d_y}{2}) \forall k \in [0; r_{max}]$

Konnektivitätsbedingung Die Segmentierung mit dem konventionellen k-Means-Algorithmus hat die unerwünschte Folge, dass die Regionen, die man als Segmentierungsergebnis erhält, in der Bildebene eventuell nicht zusammenhängend sind (siehe Abb. 4.5 links). Das liegt daran, dass eine Region zwar aufgrund der Nächster-Nachbar-Klassifikation im Merkmalraum zusammenhängt, dies aber keineswegs zur Folge hat, dass diese Bereiche im Bild ebenfalls zusammenhängen. Genau dies ist aber erwünscht, man muss also versuchen, eine beliebig geformte Region in mehrere zusammenhängende neue Regionen aufzuspalten. Dieses Verfahren ist in Abb. 4.5 exemplarisch dargestellt: Die zusammenhängenden oberen Regionen 1, 2 und 3 behalten ihren Bezeichner, da sie von oben nach unten gesehen zum ersten Mal auftauchen, für die erneuten Vorkommen von 2 und 3 am unteren Bildrand werden die neuen Bezeichner 5 und 6 vergeben. Die Region 4 ist zwar zusammenhängend, wird aber aufgelöst, da sie einen Grenzwert für die minimale Regionengröße unterschreitet, für den hier 100 Pixel angesetzt werden. Der Bezeichner 4 wird aber nicht neu vergeben, da dieser Algorithmus auch das Aussterben und Wiederauftauchen von Regionen unterstützen soll. Für die Intra-Frame-Segmentierung gilt aber: Da das Aussterben einer Region wa'ahrend der Intra-Frame-Segmentierung nicht

sinnvoll ist, werden am Ende der Intra-Segmentierung alle Bezeichner neu vergeben und somit ausgestorbene Bezeichner neu vergeben (s. *Neuzuwqeisung der Regionendeskriptor* in Kapitel 4.2.3) Dies ist aber bei der Intra-Frame-Segmentierung noch nicht von Bedeutung. Zur Erkennung zusammenhängender Bereiche wird ein rekursiv implementierter Four-Connectivity-Algorithmus verwendet [2].

Berechnung neuer Zentroide Nachdem die Vergabe neuer Bezeichner unter Berücksichtigung der Konnektivität abgeschlossen ist, werden neue Zentroide berechnet, indem die Eigenschaftsvektoren über die neuen, zusammenhängenden Bereiche gemittelt werden. Sobald sich durch die Herstellung der Konnektivität die Anzahl der Regionen nicht mehr ändert und die Schwerpunkte ihre Position nur noch geringfügig ändern, terminiert der Algorithmus, ansonsten werden die Pixel wieder mittels Nächster-Nachbar den neuen Zentroiden zugeordnet. Hierdurch geht allerdings unter Umständen die Konnektivität verloren, weshalb diese wiederhergestellt werden muss. Es ist ratsam, die Maximalzahl an solchen Iterationen zu begrenzen, da es möglich ist, dass der Algorithmus viele Iterationen benötigt, obwohl kaum mehr Änderungen feststellbar sind.

Algorithmus

1. Rechne die Koordinaten aller Zentroide auf Pixelkoordinaten um: $\overline{\mathbf{S}}_k^0 = \overline{\mathbf{S}}_k^0 \cdot (d_x, d_y) + (\frac{d_x}{2}, \frac{d_y}{2})$
   und setze Anzahl der Iterationen n_{iter} auf 0
2. Für alle Pixel $\mathbf{x}_i$:
   Füge $\mathbf{x}_i$ der Region r_k^0 mit Zentroiden $\mathbf{z}_k$: $k = \arg\min_j\{\mathrm{dist}_{kmcc}(\mathbf{x}_i, \mathbf{s}_j^0)\}$
3. Stelle Konnektivität auf der Segmentierungsmaske her
   Für jede zusammenhängende Region r_k^0:
   (a) Setze Menge der bearbeiteten Regionen-Bezeichner $\mathcal{M}_b$ auf $\emptyset$ und Anzahl zu kleiner Regionen r_{klein} auf 0
   (b) Berechne Fläche F_k^0:
   - Falls $F_k^0 > 100 \vee k \notin \mathcal{M}_b$: Füge k zu der Menge $\mathcal{M}_b$ hinzu
   - Falls $F_k^0 > 100 \vee k \in \mathcal{M}_b$: Bezeichner k schon vergeben. Zusammenhängende Region erhält neuen Bezeichner r_{max}; erhöhe r_{max} um 1
   - Falls $F_k^0 < 100$: Zusammenhängende Region erhält neuen negativen Bezeichner $-r_{klein} - 1$; erhöhe r_{klein} um 1

   (c) Für alle kleinen Regionen r_k^0 mit $k < 0$:
   - Ermittle den Zentroiden $\mathbf{z}_k$
   - Ermittle alle gültigen Nachbarregonen r_n^0 und deren Zentroide $\mathbf{z}_n$
   - Ermittle Nachbarzentroiden $\mathbf{z}_m$ mit geringster Distanz zu $\mathbf{z}_k$
   - Füge r_k^0 zu $\mathbf{s}_m^0$ hinzu
4. Sichere alte Zentroiden: $z_{k,alt} = z_k \forall k \in [0; z_{max}]$
5. Berechne Zentroiden neu:

$$\mathbf{z_k} = (\overline{\mathbf{I}}_k^0, \overline{\mathbf{V}}_k^0, \overline{\mathbf{S}}_k^0) = (\frac{\sum_{\mathbf{x}\in r_k^0} \mathbf{I^0}(\mathbf{x})}{F_k^0}, \frac{\sum_{\mathbf{x}\in r_k^0} \mathbf{V^0}(\mathbf{x})}{F_k^0}, \frac{\sum_{\mathbf{x}\in r_k^0} \mathbf{x}}{F_k^0})$$

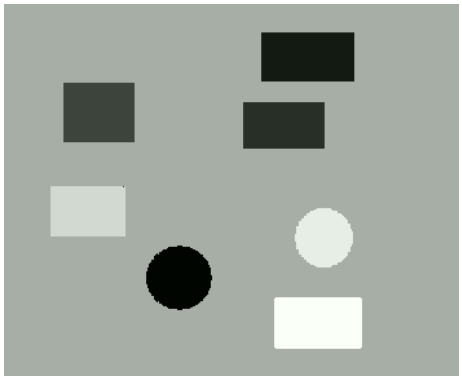

Abbildung 4.6: Segmentierung der synthetischen Sequenz nach der ersten KMCC-Iteration

6. Erhöhe n_{iter} um 1
7. Falls $n_{iter} < n_{iter,max}$ und Summe der Distanzen zwischen allen Zentroiden und dem nächsten Zentroiden aus vorheriger Iteration größer als Grenzwert wiederhole ab Schritt 2

Ergebnis Abbildung 4.6 zeigt die Segmentierung der synthetischen Sequenz nach der ersten Iteration. Die kleinen Regionen, die noch in Abb. 4.4 zu sehen waren, sind aufgelöst und die gleichfarbigen Objekte im unteren Bildbereich wurden verschiedenen Regionen zugeordnet. Diese Abbildung entspricht bereits der endgültigen Intra-Frame-Segmentierung, da in diesem einfachen Fall keine weiteren Optimierungen durchgeführt werden. Anders verhält es sich mit der zweiten Sequenz [6].

Als Beispiel für eine realistische Sequenz wird Abb. 4.7 verwendet: Links oben ist das Originalbild zu sehen, rechts oben sehen wir das Ergebnis der Vorsegmentierung.

4.2.3 Verbesserung der Segmentierungsmaske

Die Verbesserung der Segmentierungsmaske ist der letzte Schritt der Intra-Frame-Segmentierung. Sie besteht aus drei Schritten: Zunächst werden alle Objekte, die eine große farbliche Ähnlichkeit mit einer ihrer Nachbarregionen aufweisen, mit eben diesen Regionen verschmolzen und somit die Anzahl an Regionen reduziert. Anschließend werden alle Pixel, die sich in der Nähe einer Regionengrenze befinden, als *umstritten* markiert. Für jeden umstrittenen Pixel wird ermittelt, zu welcher der angrenzenden Region dieser am besten passt, zu der er dann hinzugefügt wird. Darauf muss die Konnektivität wiederhergestellt werden und abschließend wird die Anzahl der vorhandenen Regionen auf die Minimalzahl reduziert.

4.2.3.1 Verschmelzung ähnlicher Regionen

Für die Verschmelzung ähnlicher Regionen werden zunächst die Farbschwerpunkte der Regionen $\overline{S}(r_k^0)$ berechnet. Anschließend wird für jede Region die Distanz zwischen dem Farbschwerpunkt und den Schwerpunkten der Nachbarregionen ermittelt. Unterschreitet dieser Abstand einen Schwellwert, so wird die entsprechende Region als *zu verschmelzend* gekennzeichnet.

Abbildung 4.7: Einzelbild einer Videosequenz; Segmentierungsmaske vor erster KMCC-Iteration (vergrößert); nach erster Iteration (vergrößert); nach dritter Iteration

Algorithmus

1. Ermittle die Farbschwerpunkte $\overline{S}(r_k^0)$ und die Menge der Nachbarregionen g_k^0 für alle Regionen r_k^0
2. Für alle Regionen r_k^0
   (a) Ermittle die Distanz $d_F(r_k^0, r_j^0) = |\overline{S}(r_k^0) - \overline{S}(r_j^0)|$ zwischen $\overline{S}(r_k^0)$ und $\overline{S}(r_j^0)$ mit $j \in g_k$
   (b) Für alle j mit $d_F(r_k^0, r_j^0) < S_{th}$: Verschmelze Regionen r_k^0 und r_j^0

4.2.3.2 Verbesserung der Regionengrenzen

Die Verbesserung der Regionengrenzen dient dazu, ausgefranste Regionengrenzen zu glätten und die Regionengrenzen genauer den tatsächlichen Objektkonturen anzupassen. Dazu wird jeder Pixel, der in seiner Nachbarschaft eine Regionengrenze hat, als umstritten markiert. Diese umstrittenen Pixel werden anschließend der Regionen aus ihrer Nachbarschaft zugewiesen, zu der sie farblich am besten passen. Diese Entscheidung wird mittels der normalisierten Histogramme der benachbarten Regionen und der Farbinformation des umstrittenen Pixels gefällt.

Algorithmus

1. Für alle Bildpunkte $\mathbf{x} = (x, y)$ mit

$$\exists \mathbf{x_n} : R^0(\mathbf{x}) \neq R^0(\mathbf{x_n})$$

$$\mathbf{x_n} \in \{(x-1, y-1), (x, y-1), (x+1, y-1), (x-1, y),$$
$$(x+1, y), (x-1, y+1), (x, y+1), (x+1, y+1)\}$$

   R^0 ist die Segmentierungsmaske zum Zeitpunkt 0
   - $R^0(\mathbf{x}) = -1 - \min_{\mathbf{x_n}} \{R^0(\mathbf{x_n})\}$, d.h. x wird als umstritten markiert
   - Umstrittene Pixel bilden umstrittene Regionen r_k^0 mit $k < 0$
2. Für alle umstrittenen Regionen r_k^0 mit $k < 0$: Ermittle Menge g_k aller gültigen Nachbarregionen r_j^0 mit $j >= 0$
3. Für alle gültigen Regionen r_j^0 mit $j >= 0$: Errechne normalisierte Histogramme für die Farbkomponenten L,a,b: $\text{hist}_{j,L}^0(\mathbf{I}_{0,L}(\mathbf{x}))$, $\text{hist}_{j,a}^0(\mathbf{I}_{0,a}(\mathbf{x}))$ und $\text{hist}_{j,b}^0(\mathbf{I}_{0,b}(\mathbf{x}))$
4. Für jedes umstrittene Pixel $\mathbf{x}_u$ aus Region r_k^0 mit $k < 0$: Füge zu Region r_j^0 mit $\arg\min_{j \in g_k} = \{\prod_{f \in L,a,b} \text{hist}_{j,f}^0(\mathbf{I}_{0,f}(\mathbf{x}_u))\}$ hinzu
5. Stelle Konnektivität wieder her

4.2.3.3 Neuzuweisung der Regionendeskriptoren

Es ist möglich, dass im Laufe der Intra-Frame-Segmentierung Regionen gefunden wurden, die in einem der Folge-Schritte wieder ausgestorben sind. Da der Algorithmus das Aussterben von Regionen unterstützen soll, werden diese Regionendeskriptoren nicht neu vergeben. Da es allerdings nicht sinnvoll ist, bereits nach dem ersten Einzelbild von ausgestorbenen Regionen [1] zu sprechen, werden am Ende der Intra-Frame-Segmentierung die Regionendeskriptoren neu vergeben:

Es wird ermittelt, für welche Regionenbezeicher zwischen 0 und r_{max} keine Regionen existieren und alle Regionenbezeichner werden neu vergeben, sodass alle Deskriptoren zu einer existierenden Region gehören und zwischen 0 und r_{max} liegen, wobei r_{max} auf die Anzahl vorhandener Regionen gesetzt wird.

Algorithmus

1. Setze maximale Zahl der Regionen r_{max} auf 0
2. Für alle Pixel x:
   (a) Bestimme Regionsdeskriptor für Pixel mittels Segmentierungsmaske R^0: $k = R^0(\mathbf{x})$, d.h. Pixel x liegt in Region r_k^0
   (b) Falls Zuordnung $F(k) = k_n$ existiert:
      - Weise x Region $r_{k_n}^0$ zu
   (c) Sonst:
      - Weise x der Region $r_{r_{max}}^0$ zu
      - Definiere Zuordnung $F(k) = r_{max}$
      - Erhöhe r_{max} um eins

Ergebnis Das Ergebnis der Intra-Frame-Segmentierung (s. Abb. 4.8 rechts) enthält insgesamt deutlich weniger Regionen: Zum einen sind die Bezeichner der Regionen, die im Laufe der KMCC-Segmentierung ausgestorben sind, neu vergeben worden, außerdem sind farblich ähnliche Regionen verschmolzen worden, was sich vor allem in der unteren Bildhälfte bewährt, da die Bewegungsvektoren in diesem Bereich sehr unzuverlässig sind. Durch die Reduktion der Maximalzahl an Regionen sind diese im rechten Bild klarer unterscheidbar, was aber in erster Linie ein Problem der Visualisierung und nicht der Sementierung ist. Im linken Bild unterscheiden sich bestimmte Regionen nur um wenige Graustufen.

Der Effekt der Kantenverbesserung lässt sich in der vergrößerten Version darunter erkennen: Die Objektränder sind insgesamt etwas glatter und weniger ausgefranst.

4.3 Objektverfolgung und Detektion neuer Objekte

Der folgende Teil des Algorithmus dient dazu, auf Basis der vorhergehenden Segmentierungsmaske und eines Vergleichs der Farbwerte des aktuellen und letzten Einzelbildes eine neue Segmentierungsmaske zu berechnen. Dies geschieht in drei Schritten (siehe Abb. 4.9): Es werden alle Bildbereiche als umstritten markiert,

[1] *ausgestorben* bedeutet, dass eine Region, die in einer früheren Segmetierungsmaske existiert hat, in der aktuellen Segmentierungsmaske nicht mehr zu finden ist.

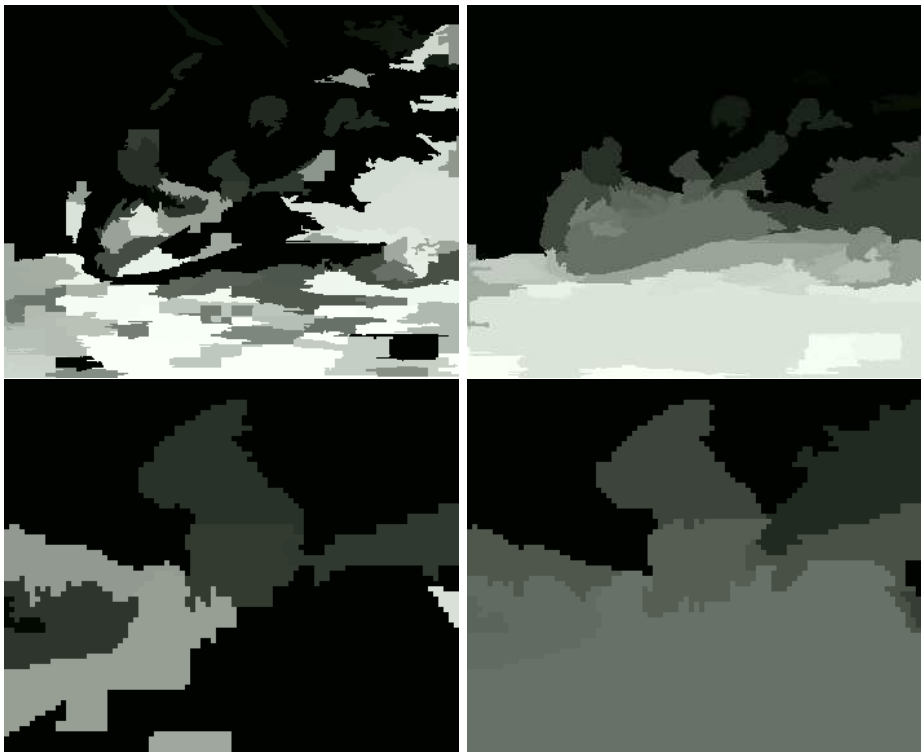

Abbildung 4.8: Oben: Segmentierungsmaske vor und nach der Maskenverbesserung; Unten: 4-fache Vergrößerung

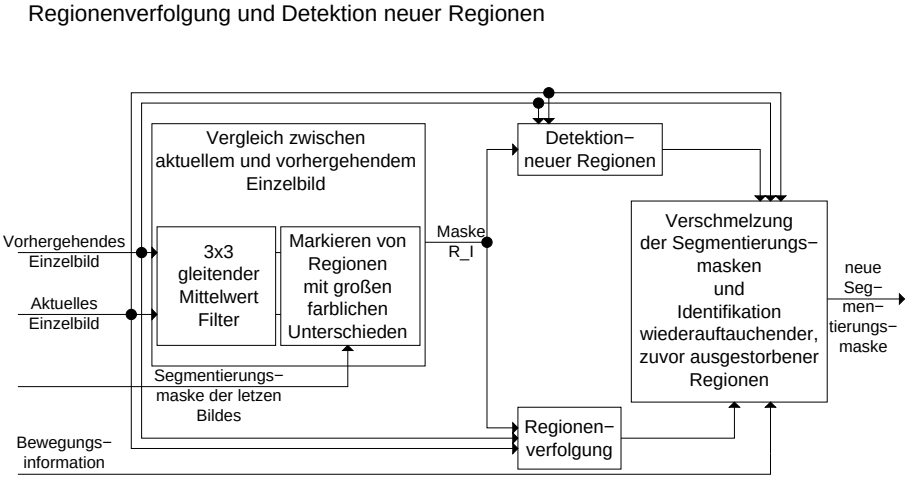

Abbildung 4.9: Übersicht zur Regionenverfolgung und Detektion neuer Regionen

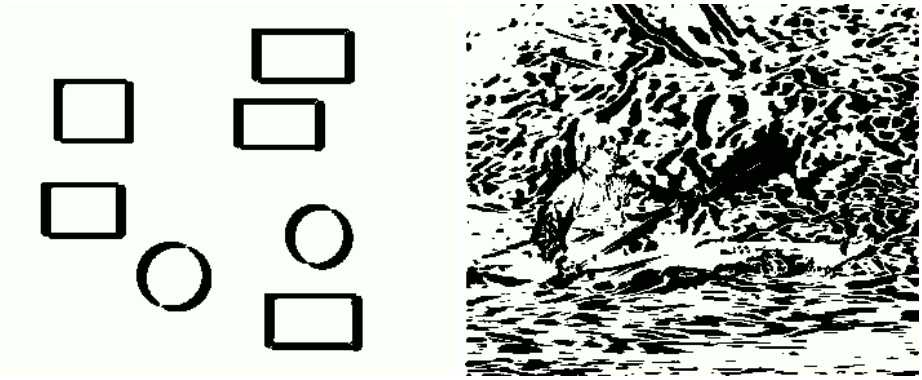

Abbildung 4.10: Beispielbilder umstrittener Bildbereiche

die starke farbliche Unterschiede zum vorherigen Bild aufweisen. Bei der *Regionenverfolgung* (s. 4.3.2) werden diese veränderten Bereiche den umliegenden Regionen angeschlossen, zu denen sie farblich am besten passen. Die *Detektion neuer Objekte* (s. 4.3.3) untersucht, ob umstrittene Bereiche potentielle neue Objekte sein können. Anschließend werden die Masken der verfolgten und der neuen Regionen verschmolzen. Mit dieser neuen Maske und der Bewegungsinformation wird anschließend bestimmt, ob bestimmte Regionen aufgrund sehr ähnlicher Regionenbewegung ein Objekt ausmachen und somit verschmolzen werden können.

4.3.1 Ermittlung umstrittener Bildbereiche

Zunächst wird sowohl auf alle Farbkomponenten im L*a*b*-Raum des aktuellen wie des vorherigen Einzelbildes ein 3x3 Pixel großer, gleitender Mittelwert-Filter h_m angewandt. Auf dem Ergebnis dieser Filterung $\widehat{\mathbf{I}}_t$ bzw. $\widehat{\mathbf{I}}_{t-1}$ wird die Farbdifferenz jedes Bildpunktes mit dem Bildpunkt aus dem vorherigen Bild ermittelt:

$$d_f(\mathbf{x}) = \|\widehat{\mathbf{I}}_t(\mathbf{x}) - \widehat{\mathbf{I}}_{t-1}(\mathbf{x})\| = \sqrt{\sum_{q\in\{L,a,b\}} (\widehat{I}_{t,q}(\mathbf{x}) - \widehat{I}_{t-1,q}(\mathbf{x}))^2}$$

Überschreitet diese Farbdifferenz einen Grenzwert $d_u = 6$, wird das Pixel als umstritten markiert, es kann also nicht gewährleistet werden, dass dieser Pixel noch zur selben Region gehört, wie im vorherigen Bild. Dann wird die Konnektivitätsbedingung auf die umstrittenen Regionen angewandt und man erhält $r_{u,max}$ umstrittene Regionen r_k^t mit $k \in [-r_{u,max}; -1]$. Abbildung 4.10 zeigt beispielhaft solche umstrittenen Bildbereiche für das zweite Einzelbild der synthetischen Sequenz (Originalbild: Abb. 3) und der Kanu-Sequenz (Originalbild Abb. 4.8). Die umstrittenen Bereiche sind schwarz. Im Falle der synthetischen Sequenz sind dies nur die Bereiche, um die sich die geometrischen Formen nach links-oben verschoben haben. Die Kanu-Sequenz weist erheblich mehr heterogene Bewegung auf, weshalb ein großer Bereich des Bildes als verändert markiert ist.

Algorithmus

1. `Wende Mittelwertfilter auf Farbkomponenten an:` $\widehat{I}_{t,q}(\mathbf{x}) = I_{t,q}(\mathbf{x}) * h_m$ `für` $q \in \{L, a, b\}$
2. `Für alle Bildpunkte x`

(a) Bestimme $d_f(\mathbf{x})$

(b) Falls $d_f(\mathbf{x}) > d_u$: Pixel $\mathbf{x}$ erhält Regionenbezeichner -1

3. Wende Konnektivitätsbedingung auf umstrittene Region r_{-1}^t an

Ergebnis: Segmentierungsmaske $R_I(\mathbf{x})$ mit gültigen Regionen r_k^t mit $k \in [0, r_{max}[$ und umstrittene Regionen r_i^t mit $i \in [-r_{u,max}; -1]$

4.3.2 Regionenverfolgung

Die Regionenverfolgung greift die Maske R_I aus Abschnitt 4.3.1 auf und schließt alle Pixel aus umstrittenen Regionen einer gültigen Nachbarregion an. Die Entscheidung, zu welcher Region ein umstrittener Pixel am besten passt, wird auf Basis der Histogramme der gültigen Nachbarregionen und den Farbwerten des Pixels gefällt. Die Regionenverfolgung nutzt lediglich Farbinformation und nicht auf Bewegungsdaten.

Für jeden Pixel $\mathbf{x}$ wird die Region gewählt, welche die Wahrscheinlichkeit $p(\mathbf{I}_t(\mathbf{x})|r_k^t)$ (mit $k \in g_j$ Menge der Nachbarregionen zu Region $r_j^t; j = R_I(\mathbf{x})$) minimiert. Für die Wahrscheinlichkeit wird vereinfacht angenommen:

$$p(\mathbf{I}_t(\mathbf{x})|r_k^t) \approx p(\mathbf{I}_t(\mathbf{x})|r_k^{t-1}) = \prod_{q \in L,a,b} \mathrm{hist}_{q,k}^{t-1}(I_{t,q}(\mathbf{x}))$$

$\mathrm{hist}_{q,k}^{t-1}(I_{t,q}(\mathbf{x}))$ bezeichnet das normalisierte Histogram für Farbkomponente q der Region r_k^{t-1}.

$$\mathrm{hist}_{q,k}^{t-1}(I_{t,q}(\mathbf{x})) = \frac{\mathrm{Hist}_{q,k}^{t-1}(I_{t,q}(\mathbf{x}))}{M_k^{t-1}}$$

mit M_k^{t-1} Größe der Region k zum Zeitpunkt t-1.

Das Histogramm der Region aus dem vorherigen Einzelbild wird deshalb verwendet, weil zum einen davon ausgegangen werden kann, dass das Histogramm einer Region über die Zeit in etwa gleich bleibt und die Region zum Zeitpunkt t aufgrund der hinzugekommenen umstrittenen Regionen so klein sein könnte, dass nicht ausreichend Werte für die zuverlässige Berechnung eines Histgramms vorliegen.

Algorithmus

1. Dupliziere Segmentierungsmaske R_I als Maske der verfolgten Regionen R_E
2. Für alle umstrittenen Regionen r_i^t mit $i \in [-r_{u,max}; -1]$:
   - Bestimme Menge g_i aller Regionenbezeichner gültiger Nachbarregionen
3. Für alle umstrittenen Pixel $\mathbf{x}_u$:
   (a) Bestimme umstrittene Region r_i^t zu der $\mathbf{x}_u$ gehört: $i = R_I(\mathbf{x_u})$
   (b) Bestimme $j = \arg\min_{k \in g_i} \left\{ \prod_{q \in L,a,b} \mathrm{hist}_k^{q,t-1}(I_{t,q}(\mathbf{x})) \right\}$
   (c) Weise $\mathbf{x}_u$ in der Segmentierungsmaske R_E der Region r_j^t zu, d.h. $R_E(\mathbf{x}_u) := j$
4. Stelle Konnektivität auf der Maske R_E her

Ergebnis: Segmentierungsmaske der verfolgten Regionen R_E

4.3.3 Detektion neuer Regionen

Ob eine umstrittene Region als neues Objekt interpretiert wird, hängt davon ab, ob diese Region bestimmte Bedingungen erfüllt, die in vier Regeln festgelegt sind:

Regel 1 - Identifikation potentiell neuer Regionen auf Basis farblicher Homogenität: Eine umstrittene Region r_j^t ist eine mögliche neue Region, wenn ihr minimaler charakteristischer Wert P_j^t deutlich größer ist, als der kleinste charakteristische Wert P_k^t aller Nachbarregionen r_k^t mit $k \in g_j$:

$$\frac{P_j^t}{\min_{k \in g_j}\{P_k^t\}} < 0.05$$

Der charakteristische Wert einer gültigen Region r_k^t ist definiert als:

$$P_k^t = E\left\{p(\mathbf{I}_t(\mathbf{x})|r_k^t)\right\} \forall \mathbf{x} \in r_k^t$$

Es handelt sich hierbei um den Mittelwert der Histogrammwerte $\text{hist}_{q,k}^{t-1}(I_{t,q}(\mathbf{x}))$ für jeden Pixel $\mathbf{x} \in r_k^t$. Ein relativ großer Wert bedeutet, dass im Mittel alle Pixel farblich besonders gut in die Region passen, also die Region eine große farbliche Homogenität aufweist. Der charakteristische Wert einer umstrittenen Region r_j^t ist definiert als:

$$P_j^t = E\left\{\max_{k \in g_j}\{p(\mathbf{I}_t(\mathbf{x})|r_k^t)\}\right\} \forall \mathbf{x} \in r_j^t$$

Ist dieser Wert groß, bedeutet dies, dass sich die Pixel der umstrittenen Region farblich sehr gut in die gültigen Nachbarregionen fügen, es sich also höchstwahrscheinlich nicht um eine neue Region handelt. Ist der Wert hingegen sehr klein, haben wir es mit Pixeln zu tun die sich farblich sehr schlecht einer anderen Region zuordnen lassen, da sie die farbliche Homogenität der Nachbarregionen beeinträchtigen würden.

Algorithmus:

1. Bestimme alle charakteristischen Werte der gültigen Regionen $P_k^t; k \in [0; r_{max} - 1]$ und ungültigen Regionen $P_j^t; j \in [-r_{u,max}, -1]$
2. Für alle ungültigen Regionen r_j^t:
   (a) Bestimme Mengen g_j der Regionendeskriptoren aller gültigen Nachbarregionen der Region r_j^t
   - Falls $\frac{P_j^t}{\min_{k \in g_j}\{P_k^t\}} < 0.05$: Füge Region r_j^t zur Menge g_{neu} potentiell neuer Regionen hinzu

Regel 2 - Pixelweise Zuordnung potentiell neuer Regionen: Für alle potentiell neuen Regionen wird nun für jeden einzelnen Pixel bestimmt, ob er farblich besser zur neuen Region oder zu einer gültigen Nachbarregion passt. Jede Position, die einer potentiell neuen Region zuzurechnen ist, wird in einer neuen, anfangs leeren Segmentierungsmaske R_N der neuen Region zugewiesen. Auf R_N wird anschließend die Konnektivitätsbedingung angewandt, wobei alle neuen Regionen, die die eine Größe von 0,2% der gesamten Bildgröße unterschreiten, aufgelöst werden. Anschließend erhält man eine Maske R_N mit zusammenhängenden neuen Regionen r_n^t mit $n \in [0; r_{n,max} - 1]$ wobei $R_N(\mathbf{x}) = -1 \forall \mathbf{x} \notin \bigcup_{n \in [0; r_{n,max}-1]} r_n^t$. Diese Maske R_N ist nun mit der Maske der verfolgten Regionen R_E zu verbinden.

Algorithmus

1. Für alle potentiell neuen Regionen r_j^t mit $j \in g_{neu}$
   - Bestimme Histogramme $\text{hist}_{L,j}^t$, $\text{hist}_{a,j}^t$ und $\text{hist}_{b,j}^t$
2. Für jeden umstrittenen Pixel $\mathbf{x}_u$ aus potentiell neuer Region r_j^t mit $j = R_I(\mathbf{x}_u)$
   - Falls $\prod_{q\in\{L,a,b\}} \text{hist}_{q,j}^t(I_{t,q}(\mathbf{x}_u)) > \prod_{q\in\{L,a,b\}} \text{hist}_{q,k}^t(I_{t,q}(\mathbf{x}_u)) \forall k \in g_j$: Setze $R_N(\mathbf{x}_u) = R_I(\mathbf{x}_u)$
3. Stelle Konnektivität auf R_N her, entferne dabei alle Regionen kleiner als 0,2% der Pixel des Bildes

Ergebnis: Segmentierungsmaske neuer Regionen R_N mit Anzahl neuer Regionen $r_{n,max}$

4.3.4 Zusammenführung der Segmentierungsmasken

Nun liegen sowohl die Maske der verfolgten Regionen R_E als auch die Maske der neuen Regionen R_N vor. Diese werden zu der endgültigen Segmentierungsmaske R^t zusammengefügt:

$$R^t(\mathbf{x}) = \begin{cases} R_E(\mathbf{x}) & \text{, falls } R_N(\mathbf{x}) = -1 \\ R_N(\mathbf{x}) + r_{max} & \text{, falls } R_N(\mathbf{x}) \neq -1 \end{cases}$$

Überall wo R_N definiert ist, erhält R^t die Regionenbezeichner aus R_N erhöht um r_{max}. Die Gesamtzahl der Regionen beläuft sich damit auf $r_{max} = r_{max} + r_{n,max}$. Es folgt die Anwendung der Regeln 3 und 4, die neue Regionen mit ähnlichen vorhandenen verschmilzt und falls angebracht, neue Regionen als wiederaufgedeckte ausgestorbene Regionen identifiziert.

Regel 3 - Verschmelzung neuer Regionen mit ähnlichen existierenden Regionen: Hat eine neue Region einen ähnlichen Farbschwerpunkt wie eine Nachbarregion wird sie mit dieser verschmolzen, da davon ausgegangen werden kann, dass diese neue Region nicht ein neu in Erscheinung getretenes Obbjekt repräsentiert, sondern eher zu der ähnlichen Nachbarregion gehört.

Algorithmus: Für alle neuen Regionen r_j^t mit $j > r_{max} - r_{n,max}$;
$(r_{max} - r_{n,max})$ ist die Anzahl der Regionen vor der Maskenverschmelzung

1. Ermittle Farbschwerpunkt $\overline{\mathbf{I}}(r_j^t)$
2. Ermittle Menge der Nachbarregionen g_j und deren Farbschwerpunkte $\overline{\mathbf{I}}(r_k^t) \forall k \in g_j$

- Falls $\min_{k\in g_j} \parallel \overline{\mathbf{I}}(r_j^t) - \overline{\mathbf{I}}(r_k^t) \parallel < C_{th}$:
  Schließe Region r_j^t der Region r_k^t mit $k = \arg\min_{k\in g_j} \parallel \overline{\mathbf{I}}(r_j^t) - \overline{\mathbf{I}}(r_k^t) \parallel$ an

Regel 4 - Identifikation neuer Regionen mit wieder aufgedeckten ausgestorbenen Regionen: Wenn ein Objekt vorübergehend durch ein örtlich davorliegendes Objekt verdeckt wird, würde die damit assozierte Region aussterben und dasselbe Objekt würde beim Wiederaufdecken mit einer neuen Region identifiziert. Ein ähnliches Phänomen tritt auf, wenn ein Objekt sich derartig schnell bewegt,

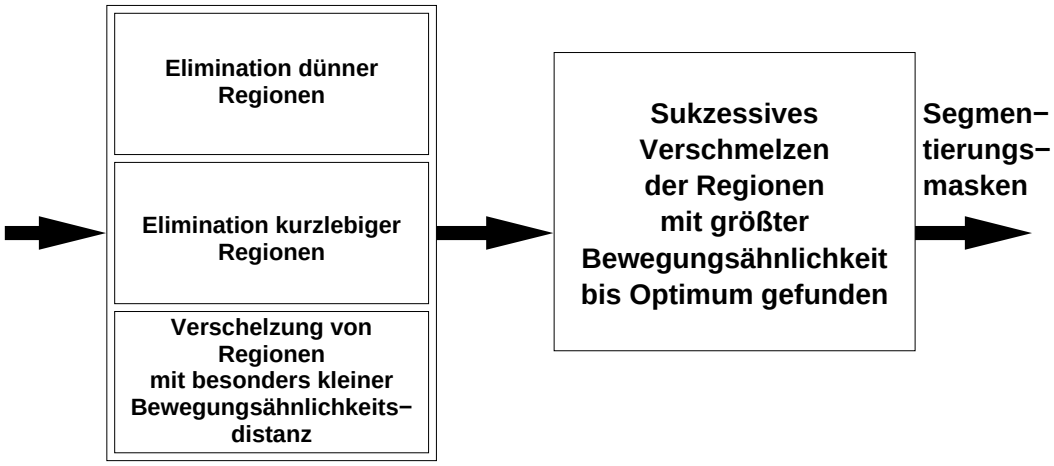

Abbildung 4.11: Übersicht zur bewegungsbasierten Regionenverschmelzung

dass die damit assoziierte Region in zwei aufeinanderfolgenden Einzelbildern keine Überschneidung aufweist. Die Regionenverfolgung würde in diesem Fall scheitern und die Region aus dem vorherigen Einzelbild würde als ausgestorben behandelt werden, während dem selben Objekt im aktuellen Einzelbild eine neue Region zugeordnet würde.

Da diese beiden Effekte unerwünscht sind, ist es erforderlich, durch *Regel 4* sicherzustellen, dass in solchen Fällen neue Regionen mit den ausgestorbenen Regionen aus vorangegangenen Einzelbildern oder der aktuellen Maske R_E der verfolgten Regionen assoziiert werden, falls eine große farbliche und örtliche Ähnlichkeit zwischen der neuen und einer der alten Regionen besteht.

Algorithmus

1. `Bestimme den Farbschwerpunkt` $\overline{\mathbf{I}}(r_j^t)$ `sowie den Ortsschwerpunkt` $\overline{\mathbf{S}}(r_j^t)$ `für jede neue Region` r_j^t `mit` $j > r_{max} - r_{n,max}$
2. `Falls`

   $$\exists k : \| \overline{\mathbf{I}}(r_j^t) - \overline{\mathbf{I}}(r_k^t) \| < C_{th} \wedge \| \overline{\mathbf{S}}(r_j^t) - \overline{\mathbf{S}}(r_k^t) \| < S_{th} \forall (k < r_{max}) \wedge (k \notin R_E)$$

   `neue Region` r_j^t `wird Region` r_k^t

Anmerkungen: Zum Anwenden von Regel vier ist es erforderlich, die letzten Orts- und Farbschwerpunkte von allen Regionen bereitzuhalten, so dass der letzte Stand der Regionenschwerpunkte festgehalten wird, sollte sie im nächsten Schritt nicht mehr existieren. Auch die folgenden Regeln werden auf derartige Informationen zurückgreifen.

Angemessene Werte für die Konstanten sind: $C_{th} = 10$; $S_{th} = 0,2\sqrt{x_{max}^2 + y_{max}^2}$

4.3.5 Bewegungsbasierte Regionenverschmelzung

Nachdem die bewegten Regionen verfolgt und neu in Erscheinung getretene Regionen erkannt wurden, gilt es nun, alle Regionen, die *räumlich-zeitliche Nachbarn* sind

und eine ähnliche *Regionen-Bewegungsrichtung* aufweisen, zu verschmelzen, da sie wahrscheinlich Teile desselben Objekts sind.

Dies ist leider erst möglich, sobald die Segmentierung der Videosequenz komplett abgeschlossen ist, da die bewegungsbasierte Regionenverschmelzung auf Indikatoren zurückgreift, die über die komplette Lebensdauer der Regionen akkumuliert werden müssen. Der Prozess der Regionenverfolgung bzw. -neudetektion muss also komplett abgeschlossen sein und während dieser Phase werden die Indikationren akkumuliert, die am Ende angeben, welche Regionen zusammen ein Objekt bilden könnten.

Doch zusächst sollen die Prinzipien der *räumlich-zeitlichen Nachbarschaft* und der *Regionen-Bewegungsrichtung* geklärt werden:

Räumlich-zeitliche Nachbarschaft: Es liegt eine räumlich-zeitliche Nachbarschaft zwischen zwei Regionen r_i, r_j vor, wenn diese Regionen immer dann aneinandergrenzen, wenn sie in einer Segmentierungsmaske R^t koexistieren:

$$\forall t : i \in R^t \vee j \in R^t \Rightarrow i \in g_j \vee j \in g_i$$

Regionen-Bewegungsrichtung: Um die Bewegungsrichtung $\begin{pmatrix} u_k \\ v_k \end{pmatrix}$ einer Region zu bestimmen, wird das bilineare Bewegungsmodell verwendet [4]:

$$u_k = a_{i,0} + a_{i,1}x + a_{i,2}y + a_{i,3}xy$$

$$v_k = a_{i,4} + a_{i,5}x + a_{i,6}y + a_{i,7}xy;$$

$$\forall x, y \in r_i^t$$

Die Parameter $a_{i,0} \ldots a_{i,7}$ werden durch die *Methode der kleinsten Quadrate* geschätzt: Es wird versucht die Summe über aller Pro-Pixel-Bewegungsvektoren $\mathbf{u_k}$ mit $k = 0, 1 \ldots N-1$ innerhalb der N Pixel großen Region r_i^t

$$\sum_{k=0}^{N-1} (u_k - (a_{i,0} + a_{i,1}x_k + a_{i,2}y_k + a_{i,3}x_k y_k))^2 + (v_k - (a_{i,0} + a_{i,1}x_k + a_{i,2}y_k + a_{i,3}x_k y_k))^2$$

zu minimieren. Dies wird durch Nullsetzen der Ableitungen der Summen für u bzw. v nach den Modellparametern a_0, a_1, a_2, a_3 bzw. a_4, a_5, a_6, a_7 erreicht. Dies liefert für u und v jeweils vier Gleichungen mit vier Modellparametern als Unbekannte. Um die Modellparameter explizit zu berechnen, sind dann folgende Gleichungssysteme zu lösen:

$$\begin{pmatrix} N & \sum_{k=0}^{N-1} x_k & \sum_{k=0}^{N-1} y_k & \sum_{k=0}^{N-1} x_k y_k \\ \sum_{k=0}^{N-1} x_k & \sum_{k=0}^{N-1} x_i^2 & \sum_{k=0}^{N-1} x_k y_k & \sum_{k=0}^{N-1} x_i^2 y_k \\ \sum_{k=0}^{N-1} y_k & \sum_{k=0}^{N-1} x_k y_k & \sum_{k=0}^{N-1} y_i^2 & \sum_{k=0}^{N-1} x_k y_i^2 \\ \sum_{k=0}^{N-1} x_k y_k & \sum_{k=0}^{N-1} x_i^2 y_k & \sum_{k=0}^{N-1} x_k y_i^2 & \sum_{k=0}^{N-1} x_i^2 y_i^2 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix} = \begin{pmatrix} \sum_{k=0}^{N-1} u_k \\ \sum_{k=0}^{N-1} u_k x_k \\ \sum_{k=0}^{N-1} u_k y_k \\ \sum_{k=0}^{N-1} u_k x_k y_k \end{pmatrix}$$

$$\begin{pmatrix} N & \sum_{k=0}^{N-1} x_k & \sum_{k=0}^{N-1} y_k & \sum_{k=0}^{N-1} x_k y_k \\ \sum_{k=0}^{N-1} x_k & \sum_{k=0}^{N-1} x_i^2 & \sum_{k=0}^{N-1} x_k y_k & \sum_{k=0}^{N-1} x_i^2 y_k \\ \sum_{k=0}^{N-1} y_k & \sum_{k=0}^{N-1} x_k y_k & \sum_{k=0}^{N-1} y_i^2 & \sum_{k=0}^{N-1} x_k y_i^2 \\ \sum_{k=0}^{N-1} x_k y_k & \sum_{k=0}^{N-1} x_i^2 y_k & \sum_{k=0}^{N-1} x_k y_i^2 & \sum_{k=0}^{N-1} x_i^2 y_i^2 \end{pmatrix} \begin{pmatrix} a_4 \\ a_5 \\ a_6 \\ a_7 \end{pmatrix} = \begin{pmatrix} \sum_{k=0}^{N-1} v_k \\ \sum_{k=0}^{N-1} v_k x_k \\ \sum_{k=0}^{N-1} v_k y_k \\ \sum_{k=0}^{N-1} v_k x_k y_k \end{pmatrix}$$

Anschließend wird der mittlere Schätzfehler auf Basis der errechneten Modellparameter berechnet:

$$\overline{E} = \frac{1}{N} \sum_{k=0}^{N-1} \sqrt{(u_k - \widehat{u}_k)^2 + (v_k - \widehat{v}_k)^2}$$

$\widehat{v}_k$ bzw. $\widehat{u}_k$ sind die mit den oben ermittelten Modellparametern berechneten Schätzwerte für die Regionenbewegung. u_k, v_k die Pro-Pixel-Bewegungsrichtungen innerhalb der Region.

Anschließend wird für jeden Pro-Pixel-Bewegugnsvektor die Abweichung vom Schätzvektor berechnet. Ist diese größer als E_{avg} wird der Bewegungsvektor als zurückgewiesen markiert und erneut die Modellparamerter auf Basis der nicht-zurückgewiesenen Vektoren berechnet. Dies wird so lange wiederholt, bis keine Bewegungsvektoren mehr zurückgewiesen werden. Dieses Zurückweisen bestimmter Bewegugsvektoren dient dazu, durch die Bewegungsschätzung bedingt potentiell fehlerhafte Bewegungsvektoren zu erkennen und zu eliminieren.

Algorithmus zur Berechnung der Modell-Parameter einer Region oder der Vereinigung zweier Regionen

- Markiere alle Pro-Pixel-Bewegungsvektoren als $\mathbf{u}_k$ als nicht zurückgewiesen
- Für alle Pixel $\mathbf{x_k}$ mit Bewegungsvektoren $\mathbf{u}_k^t$ in Region r_i^t bzw. Vereinigung zweier Regionen $r_i^t = r_m^t \bigcup r_n^t$

1. Ermittle die Modellparameter $a_{i,0} \dots a_{i,7}$ unter Verwendung aller nicht-zurückgewiesenen Bewegungsvektoren
2. Berechne unter Verwendung der Modellparameter den mittleren Fehler $\overline{E}_i = \frac{1}{M_i} \sum_{k=0}^{M_k - 1} \sqrt{(u_k - \widehat{u}_k)^2 + (v_k - \widehat{v}_k)^2}$
3. Berechne die Abweichung jedes Bewegungsvektors $\mathbf{u}_k$ vom jeweiligen geschätzen Bewegungsvektor der zugehörgen Region r_i^t: $E_{i,k} = \sqrt{(u_k - \widehat{u}_k)^2 + (v_k - \widehat{v}_k)^2}$
   - Falls $E_{i,k} > \overline{E}_i$: Markiere $\mathbf{u}_k$ als zurückgewiesen
4. Falls Bewegungsvektoren zurückgewiesen wurden: Wiederhole ab Schritt 1

Ergebnis: Vektor der Modellparameter $\mathbf{U}^t(s_i) = (a_{i,0}, a_{i,1} \dots a_{i,7})^T$; Summe der quadrierten Bewegungskompensationsfehler $\overline{E}_i$

Bewegungsähnlichkeitsmaß zwischen Regionen: Die Bewegungsähnlichkeit von räumlich-zeitlichen Nachbarregionen r_m, r_n wird über folgende Formel bestimmt:

$$D_U(r_m, r_n) = \frac{\sum_{t=1}^{T-1} \Upsilon_t(r_m)\Upsilon_t(r_n) D_U^t(r_m, r_n)}{\max\{1, \sum_{t=1}^{T-1} \Upsilon_t(r_m)\Upsilon_t(r_n)\}}$$

T ist die Länge der Sequenz; $\Upsilon_t(s_m) = 1$ falls $r_m \bigcap R^t \neq \emptyset$, 0 sonst; D_U^t ist die Bewegungsähnlichkeit der Regionen zur Zeit t.

Der Nenner entspricht der Anzahl an Einzelbildern, in denen beide Regionen existieren. Dieser Wert ist immer mindestens 1, da die Bewegungsähnlichkeit nur für räumlich-zeitliche Nachbarn interessant ist und diese immer mindestens in einem Bild koexistieren. Die Bewegungsähnlichkeit zum Zeitpunkt t ist folgendermaßen definiert:

$$D_U^t(r_m, r_n) = \frac{E_{m,n}^t(r_m) - E_m^t(r_m)}{M_m^t} + \frac{E_{m,n}(r_n) - E_n^t(r_n)}{M_n^t}$$

$E_m(r_m)$ ist der mittlere Fehler der Bewegungskompensation für die Region r_m (s. Algorithmus zur Berechnung der Modellparameter in Kapitel 4.3.5 auf S. 31); $E^t_{m,n}(r_m)$ ist der Fehler der Bewegungskompensation für Region r_m mit den Modellparametern, die aus der Vereinigung der Regionen $r_m \bigcup r_n$ ermittelt wurden. M^t_m ist die Größe der Region r_m zum Zeitpunkt t.

Das Bewegungsähnlichkeitsmaß kann inkrementell für alle räumlich-zeitlichen Nachbarn berechnet werden, indem nach der Segmentierung eines Einzelbildes für alle räumlich-zeilichen Nachbarn die beide in der Segmentierungsmaske R^t vorkommen ($r_m \cap R^t \neq \emptyset \vee r_n \cap R^t \neq \emptyset$) der Zähler des Bruches um $D^t_U(r_n, r_m)$ inkrementiert wird. Sollte die räumlich-zeitliche Nachbarschaft zwischen den Regionen nicht mehr gegeben sein, ist $D_U(s_m, s_n)$ irrelevant und r_n kann aus der Liste der räumlich-zeitlichen Nachbarn von r_m entfernt werden (und umgekehrt). Auf diese Weise steht nach Abschluss der Sementierung die Information über die räumlich-zeitliche Nachbarschaft und die zugehörigen Bewegugsähnlichkeiten bereit und die bewegungsbasierte Regionenverschmelzung kann durchgeführt werden.

Regionenverschmelzung Die bewegungsbasierte Regionenverschmelzung (siehe Abb. 4.11) basiert darauf, solange die räumlich-zeitlichen Nachbarregionen zu verschmelzen, bis das Optimum an Regionen erreicht ist. Dabei werden zunächst Regionen verschmolzen, die die Bedingungen folgender Regeln erfüllen:

Regel 5 - Elimination kurzlebiger Regionen:
Existiert eine Region r_m insgesamt weniger als vier Einzelbilder lang, d.h. $\sum_{t=1}^{T} \Upsilon_t(r_m) < 4$, wird diese Region mit dem räumlich-zeitlichen Nachbarn r_n mit kleinster Distanz $D_U(r_m, r_n)$ verschmolzen ($n = \arg\min_i D_U(s_m, s_i)$).

Regel 6 - Elimination schmaler Regionen:
Ist eine Region ungewöhnlich schmal, wird davon ausgegangen, dass diese Region allein kein tatsächliches Objekt repräsentieren kann — sie muss also mit einem räumlich-zeitlichen Nachbarn verschmolzen werden. Die Bedingung, die besagt, dass eine Region r_m zu schmal ist, lautet folgendermaßen:

$$\sum_{t=1}^{T} \frac{M^t_m}{\sqrt{{bb^t_{m,x}}^2 + {bb^t_{m,y}}^2}} < 6 \cdot \sum_{t=1}^{t} \Upsilon_t(r_m)$$

Eine Region, die diese Bedingung erfüllt, wird mit der Region r_n verschmolzen, die folgende Bedingungen erfüllt und zugleich $D_U(r_m, r_n)$ minimiert.

$$\sum_{t=1}^{T} \frac{M^t_n}{\sqrt{{bb^t_{n,x}}^2 + {bb^t_{n,y}}^2}} \geq 6 \sum_{t=1}^{t} \Upsilon_t(r_n)$$

und

$$\frac{\sum_{t=1}^{t} \sqrt{{bb^t_{n,x}}^2 + {bb^t_{n,y}}^2}}{\sum_{t=1}^{t} \Upsilon_t(r_n)} \geq \frac{\sum_{t=1}^{t} \sqrt{{bb^t_{m,x}}^2 + {bb^t_{m,y}}^2}}{\sum_{t=1}^{t} \Upsilon_t(r_m)}$$

$bb^t_{n,x}, bb^t_{n,y}$ sind die Breite bzw. Höhe des umrandenden Rechtecks.

Erfüllt keiner der räumlich-zeitlichen Nachbarn diese Bedingungen, wird zunächst die obere fallengelassen. Sollte sich immer noch kein geeigneter Kandidat finden, wird auch die untere Bedingung gestrichen, d.h. man wählt einfach die Region, die $D_U(r_m, r_n)$ minimiert. Nachdem die entsprechenden zu verschmelzenden Regionen ermittelt wurden, werden diese Verschmelzungen in allen Segmentierungsmasken R^t mit $t \in [0; T-1]$ vollzogen und dabei die Parameter $D_U(r^t_m)$ und $\Upsilon_t(r_m)$ für alle Regionen r_m aktualisiert.

Ist dies abgeschlossen, wählt man die räumlich-zeitlichen Nachbarn r_m, r_n , für welche $D_U(r_m, r_n)$ minimal ist. Dies wird so lange wiederholt, bis das Verhältnis der minimalen räumlich-zeitlichen Abstände $D_{U,k} = min_{m,n}\{D_U(r_m, r_n)\}$ ein globales Maximum erreicht hat:

$$\frac{D_{U,k}}{D_{U,k+1}}, \forall k \in [K-2, 1], D_{U,k+1} > 0; D_{U,k} > D_{U,min}$$

k ist die Anzahl der Regionen nach der Verschmelzung der Regionen mit minimalem Abstand $D_{U,k}$; $D_{U,min}$ ist ein Schwellwert, dessen Unterschreiten durch $D_{U,k}$ auf jeden Fall zu einer Verschmelzung führen soll, z.B. $D_{U,min} = 0,1$. Als endgültige Segmentierung wird die Segmentierung gewählt, welche genau einen Verschmelzungsschritt vor dem globalen Maximum des Schätzfehlerverhältnisses liegt. Da es jadoch nicht möglich ist, festzustellen, ob ein lokales Maximum gleichzeitig ein globales ist, ist es erforderlich, alle Regionen sukzessive zu verschmelzen und dabei immer den Stand vor dem höchsten lokalen Maximum zu sichern. Sind alle Regionen verschmolzen, wird dieser Stand wiederhergestellt und stellt somit die finale Segmentierung dar.

4.4 Abweichungen vom und Ergänzungen des ursprünglichen Algorithmus

In einigen Fällen lässt [1] Details des vorgestellten Algorithmus offen oder es wird in der Implementierung in eingigen Fällen bewusst von den vorgeschlagenen Methoden abgewichen. Dies ist auch der Grund dafür, warum dieses Dokument den Umfang der ursprünglichen Dokumentation des Algorithmus deutlich überschreitet. Die wichtigsten Unterschiede und Ergänzungen sollen hier kurz angeführt werden:

Der Maximin- und der K-Means-Algorithmus aus der Vorsegmentierung verwendet ursprünglich nicht die Blockgröße aus der Bewegungsschätzung sondern deutlich größere Blocke. Da die Vorsegmentierung allerdings nur einen verschwindend geringen Teil der gesamten Rechenzeit der Intra-Frame-Segmentierung in Anspruch nimmt, wurde aus Gründen der Einfachheit dieser Weg gewählt. Auf die Distanzmaße für Maximin- und K-Means wird in der ursprünglichen Quelle nicht eingegangen, weshalb hierfür angepasste Varianten der KMCC-Distanz verwendet werden. Der im Paper als *Four-Connectivity Component Labelling* benannte Algorithmus wird nicht näher erläutert und weißt in der Variante aus [2] gravierende Nachteile auf, namentlich eine große Ineffizienz und eine erhebliche Rekursionstiefe in ungünstigen Fällen, zudem ist diese Variante in dieser Form nicht zu gebrauchen, da sie nicht gewährleistet, dass vorhandene Labels nach Möglichkeit beibehalten werden. Der in der Implementierung verwendete Algorithmus beruht auf einer gemischt rekursiv bzw. iterativ implementierten Variante des in der Computergrafik gebräuchlichen Seed-Fill-Algorithmus zum Einfärben farblich einheitlich umrandeter Bildbereiche. Für die Anwendung dieser Routine zur Herstellung der Konnektivität auf der Maske R^t unmittelbar nach deren Erzeugung aus R^t_N und R^t_E wurde eine zusätzliche Erweiterung vorgenommen: Der Algorithmus zeigt hier die oftmals unerwünschte Tendenz, neue Regionen zu erzeugen. Dies beruht darauf, dass einzelne, falsch zugewiesene Pixel aus umstrittenen Bildbereichen eine ursprünglich gültige Region derart aufteilen können, dass diese nicht mehr zusammenhängend ist und somit teilweise einen neuen Bezeichner erhalten würde. Die falsch zugewiesenen Pixel werden später aber der aufgespaltenen Region zugeteilt. Deshalb wird zunächst die Konnektivitätsroutine normal angewandt, anschließend aber allen durch Aufteilung alter Regionen neu entstandenen Regionen zunächst das Label der Region zugewiesen,

die vorher die größte Fläche auf dem Gebiet der neuen Region eingenommen hat. Wird nun die Konnektivitätsroutine erneut angewandt, kann man in vielen Fällen eine Reduktion der Anzahl neugebildeter Regionen beobachten.

Die Segmentierungsmaskenverbesserung (siehe 4.2.3) ist zwar in der ursprünglichen Beschreibung erwähnt, aber nicht näher erläutert. Die in dieser Arbeit verwendete Variante beruht deshalb auf empirisch ermittelten Verfahren, die teilweise auf den Methoden aus der Regionenverfolgung (siehe 4.3.2) beruhen.

Die Vorschrift zu Bildung der Maske R_I erwies sie in der Praxis als nicht zufriedenstellend, da diese Fehler bei der Regionenverfolgung begünstigt, deshalb wird in dieser Version eine große umstrittene Region gebildet, die dann mittels Konnektivitätsbedingung in mehrere kleine Regionen aufgespalten wird (siehe 4.3.1).

Die Gleichungssysteme zur Ermittlung der Parameter des bilinearen Bewegungsmodells konnten in keiner der angegebenen oder anderen Quellen gefunden werden, sie wurden rechnerisch ermittelt. Die Lösung der Gleichungssysteme erfolgt mittels einer Gauss-Elimination aus [8]. Die Ermittlung des Bewegungsmodells soll mit Bewegungsvektoren auf Blockbasis erfolgen. In der Implementierung werden allerdings Bewegungsdaten verwendet, die auf allen Pixeln definiert sind und das Bewegungsmodell arbeitet auf Pixelbasis.

Die räumlich-zeitliche Nachbarschaft und die Indikatoren für die Bewegungsähnlichkeitsdistanz (siehe 4.3.5) werden inkrementell und nur für zu einem bestimmten Zeitpunkt potentielle räumlich-zeitliche Nachbarn ermittelt.

Kapitel 5

Ergebnisse

Abbildung 5.1 zeigt eine Gegenüberstellung der ersten sechs Einzelbilder der Sequenz Table Tennis und die dazugehörigen Segmentierungsmasken, anhand derer sich exemplarisch einige Besonderheiten des Algorithmus aufzeigen lassen.

Es lässt sich feststellen, dass der Tischtennisball zwar korrekt verfolgt wird, aber in der Segmentierungsmaske zu klein erscheit. Dieser Effekt ist eventuell darauf zurückzuführen, dass der Ball im Video aufgrund der Bewegungsunschärfe relativ verwischt erscheint und die Detektion umstrittener Bereiche auf dem Vergleich von Bildern basiert, die mit einem 3x3-Mittelwertfilter geglättet wurden. In diesem Fall werden also zu wenige Pixel als umstritten markiert. Jedoch gilt es bei dem entsprechenden Schwellwert gleichzeitig darauf zu achten, dass nicht mehr Pixel als erforderlich als umstritten markiert werden, da dies sonst andere unerwünschte Effekte zeitigen kann, zum Beispiel eine erhöhte Anzahl an falsch zugewiesenden Pixeln bei der Regionenverfolgung. Desweiteren wird die Hand anfangs nicht richtig segmentiert. Die Ursache hierfür ist, dass sich diese farblich nicht allzusehr vom Hintergrund unterscheidet und aufgrund der geringen Größe in diesem Bereich nicht ausreichend einheitliche Bewegungsvektoren vorliegen. Die Tatsache, dass der Arm in mehrere Regionen aufgeteilt ist, lässt sich vor allem auf Schattenwurf am Ärmel und abweichende Bewegungsvektoren am Bildrand zurückführen. Die abschließende bewegungsbasierte Regionenverschmelzung scheitert hier, da der gesamte Arm keine einheitliche Bewegungsrichtung aufweist.

Der Hintergrund wird korrekt erkannt, wobei die Tischtennisplatte wegen des großen farblichen Unterschiedes zum Hintergrund als eigenständiges Objekt segmentiert wird. Die Linie wird dabei zwar zunächst als einzelne Region erkannt, in der bewegungsbasierten Regionenverschmelzung jedoch mit der Platte verschmolzen, da die Linie zu dünn ist, um ein eigenständiges Objekt darzustellen.

Der Algorithmus zeigt die Tendenz, im Verlauf der Segmentierung neue Regionen zu detektieren, was nicht darauf zurückzuführen ist, dass tatsächlich *neue* Objekte entdeckt werden, sondern vorhandene Regionen ihre Konnektivität verlieren. Dies kann zum Beispiel im vierten Einzelbild aus 5.1 an der Hand beobachtet werden. Diesem Effekt wird, sofern eine solche Aufteilung nicht tatsächlich im Video vorliegt, sondern auf fehlerhafte Segmentierungsergebnisse zurückzuführen ist, zwar entgegengewirkt, z.B. durch die in 4.4 beschriebene Maßnahme oder durch die bewegungsbasierte Regionenverschmelzung. Derartige unerwünschte Aufspaltungen von Regionen sind leider dennoch in manchen Fällen zu beobachten.

Insgesamt lässt sich feststellen, dass der Algorithmus solche Sequenzen besser seg-

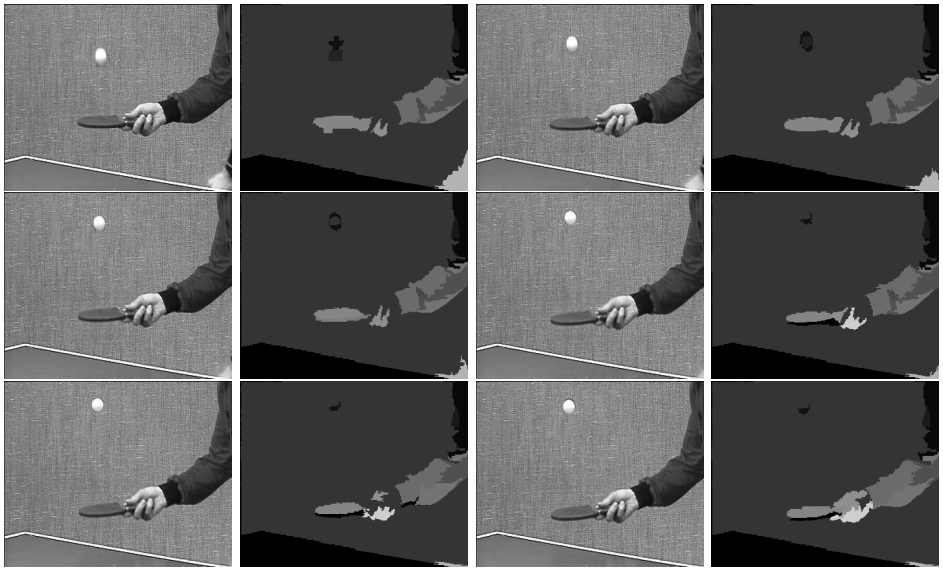

Abbildung 5.1: Segmentierung der ersten sechs Bilder der Sequenz Table-Tennis [7]

mentiert, die ingesamt einheitliche und keine extrem schnellen Bewegungen enthalten. Kleine, sehr schnell bewegte Objekte, die in zwei aufeinanderfolgenden Bilder keine überschneidung aufweisen, können nicht immer zufriedenstellend verfolgt werden, zumal in derartigen Fällen auf keine zuverlässigen Bewegungsvektoren für besagte Regionen zur Verfügung stehen. Die prinzipielle Funktionsfähigkeit der Mechanismen zur Detektion wiederaufgedeckter Regionen konnte anhand syntetischer Sequenzen nachgewiesen werden.

Kapitel 6

Zusammenfassung und Ausblick

6.1 Zusammenfassung

Mit diesem Segmentierungsalgorithmus werden viele der anfangs gestellten Anforderungen einbezogen, die von den einfachen Ansätzen nicht berücksichtigt wurden: Es werden verschiedene Merkmale genutzt, die ein Objekt ausmachen können und diese sinnvoll miteinander in Beziehung gesetzt. Dabei werden z.B. durch die Verwendung des L*a*b*-Farbraums Aspekte der menschlichen Wahrnehmung in die Segmentierung einbezogen. Eine Klasse bzw. ein Objekt stellt immer einen zusammenhängenden Bildbereich dar, es ist also nicht möglich, dass zwei getrennte Objekte der gleichen Klasse angehören. Durch die Verfolgung der Objekte auf Basis eines Farbdifferenzmaßes und nicht auf Basis eventuell unzuverlässiger Bewegungsvektoren wird erreicht, dass die Bewegung eines Objekt auch über mehrere Einzelbilder hinweg verfolgt werden kann und den gleichen Bezeichner in allen Segmentierungsmasken behält. Dies wird vor allem durch Einbeziehung von speziellen Mechanismen um Verdeckung und erneutes Erscheinen eines Objektes zu erkennen, sowie der besonderen Behandlung schnell bewegter Objekte sichergestellt. Ein aufwendiges System zur Ermittlung der Bewegungsähnlichkeit mehrerer räumlich-zeitlicher Nachbarn erlaubt die Verschmelzung von homogenen Regionen zu einem semantischen Objekt. Dabei werden unzuverlässige Bewegungsvektoren detektiert und nicht in die Berechnung einbezogen.

Letzteres birgt leider auch einen Nachteil: Das Prinzip der räumlich-zeitlichen Nachbarschaft und die Berechnung der Bewegungsähnlichkeit beruhen auf Daten, die erst nach der Durchführung eines kompletten Segmentierungsdurchlaufs zur Verfügung stehen. Erst mit Kenntnis der Segmentierung einer kompletten Sequenz kann eine Aussage darüber getroffen, welche Regionen räumlich-zeitliche Nachbarn sind und aufgrund hoher Bewegungsähnlichkeit zu einem Objekt gruppiert werden können. Es ist also nicht möglich, die komplette Segmentierung mit bewegungsbasierter Regionenverschmelzung in einem Durchlauf zu vollziehen.

6.2 Ausblick

Der Segmentierungsalgorithmus liefert nicht nur die Segmentierungsmasken für eine Videosequenz sondern ermöglicht auch die Nutzung anfallender Seiteninformationen. Die wichtigsten sind die Modellparameter für das bilineare Regionenbewegungsmodell, die die Bestimmug der Bewegungsrichtungen der Regionen erlauben. Diese können möglicherweise benutzt werden, um die Hauptbewegungsrichtungen einer Videosequenz zu ermitteln. Daraus können dann Rückschlüsse darauf gezogen werden, welchen Objekten ein Betrachter eventuell besondere Aufmerksamkeit schenkt. Diese Information kann in eine objektive Vídeoqualitätsmessung einfließen.

Der Algorithmus bietet noch einige Optimierungsmöglichkeiten bezüglich der Geschwindigkeit. Exemplarisch seien hier der Löser für die Gleichungssysteme zur Bestimmung der Modellparameter für die Regionenbewegung und der rekursiv implementierte Algorithmus zur Ermittlung zusammenhängender Regionen genannt.

Um die Szenenwechsel zu unterstützen, könnte der Algorithmus um eine Detektion von Szenenwechseln erweitert werden, die nach jedem Szenenwechsel eine erneute Intra-Frame-Segmentierung einleitet.

Das bereits implementierte bilinieare Bewegungsmodell kann leicht zu einer Detektion der Kamera-Bewegung adaptiert werden. Mit dieser Information ist es möglich die Regionenverschmelzung um eine Hintergrunddetektion zu erweitern.

Anhang A

Auflistung und Erklärung der verwendeten Funktionen

A.1 libs/motion/object_segmentation.c File Reference

Performs object segmentation.

```
#include <math.h>
#include <unistd.h>
#include "iconvert.h"
```

Data Structures

- struct **histogram_struct**
  Structure containing the normalized histograms for a single frame.
- struct **kmcc_centroid**
  Structure containing the information on the centroids for some region.
- struct **object_info_struct**
  Information which is passed to next call of object_segmentation frame by frame.
- struct **picture_info_struct**
  Structure containing the information on a single frame.
- struct **region_descriptor_list**
  Structure for one element of a linked list called region descriptor list.
- struct **region_info**
  Structure containing all information on a region.
- struct **region_motion_parameter**

Structure containing the information on the motion model for the motion of a specific region or a union of two regions.

Defines

- #define **MAX_MAXIMIN_CLASSES** 30

  Maximum classes MAXIMIN generates.

- #define **KMCC_THRESHOLD** 1

  If any component of a centroid changes its value by this threshold in the KMCC algorithm this centroids position is considered as changed.

- #define **KM_THRESHOLD** 0.001

  If any component of a centroid changes its value by this threshold in the K-Means algorithm this centroids position is considered as changed.

- #define **CONNECTIVITY_THRESHOLD** 100

  minimum size of a region in pixels for kmcc_evaluate_connectivity during first-frame-segmentation

- #define **KMCC_CENTROID_DISTANCE_THRESHOLD** 1E-6

  if the distance between the centroids changes less or equal than this value, KMCC terminates

- #define **DIF_TH** 6

  colour difference which has to be exceeded so that a pixel is considered as disputed

- #define **HISTOGRAM_ENTRIES** 10

  number of histogram bins

- #define **KMCC_CLUSTERING_MAX_ITERATIONS** 3

  maximum number of iterations KMCC performs

- #define **MASK_IMPROVEMENT_COLOR_DIST_THRESHOLD** 0.25

  colour distance for which two regions are merged during mask improvement

- #define **COLOR_DISTANCE_THRESHOLD_SEGMENTATION_MASK_FUSION_C_TH** 0.25

  colour distance for which newly detected regions are merged with neighbour regions

- #define **C_TH** 10

  colour distance which has to be underrun by a newly detected region's colour centroid so that it is identified as a newly appearing extinct region

- #define **REGION_MERGING_MOTION_SIMILARITY_THRESHOLD** 0.01

  motion similarity distance which has to be underrun so that a region is merged with a spatiotemporal neighbour in the initial merging procedure

- #define **MOTION_SIMILARITY_WITH_REJECTED_PIXELS** 0

  switch to toggle the use of the iterative rejection scheme while computing motion model parameters

- #define **FOPEN**(fileptr, filename, mode) {fileptr=fopen(filename,mode); if(fileptr==NULL) {dbg_out(DBG_ERROR,"Could not open file %s in line %ld\n",filename,__LINE__); perror(""); }}

Enumerations

- enum **region_status** { **EXISTING** = 0, **NEW**, **EXTINCT**, **MERGED** }

Functions

- double **sqr_object** (double x)

  Computes the square of a floating point number.

- double **euclid_distance** (double *vec1, double *vec2, long dim)

  Computes the euclid distance between two vectors.

- void **region_descriptor_list_init** (struct **region_descriptor_list** *list)

  Initializes a region descriptor list.

- **region_descriptor_list** * **region_descriptor_list_append** (struct **region_descriptor_list** *list_head, long value, long list_id)

  Appends an element at the end of the list if not already present in list.

- int **region_descriptor_list_is_in** (struct **region_descriptor_list** *list, long value)

  Returns TRUE if element "value" is in list, FALSE elsewise.

- **region_descriptor_list** * **region_descriptor_list_delete_first** (struct **region_descriptor_list** *list)

  Deletes first element of list.

- double **distance_maximin** (double *pix_color, double *region_color, double *pix_motion, double *region_motion, long pix_x, long pix_y, long region_x, long region_y)

  Computes the distance between a feature vector and a region center w.r.t.

- double **distance_kmcc** (double *pix_color, double *region_color, double *pix_motion, double *region_motion, long pix_x, long pix_y, long region_x, long region_y, long region_area, double average_region_area, double DI_-max, long max_block_displacement_x, long max_block_displacement_y, long width, long height)

  Computes the distance between a feature vector and a region center w.r.t.

- int **copy_file** (char *src_filename, char *dst_filename)

  This function copies a file on the disc.

- double **estimate_DImax** (struct **picture_info_struct** *picture_info)

  Computes an estimation of the image contrast; necessary for distance measure.

- void **get_pixel_neighbour_regions** (long pixel_neighbour_regions[8], long x, long y, long width, long *feature_classes)

  Finds any region descriptor of the 8 neighbouring pixels.

- void **get_valid_neighbours_of_disputed_regions** (long width, long height, long *feature_classes, struct **region_descriptor_list** **neighbours)

  Finds every valid neighbour region for every disputed region in a segmentation mask.

- void **get_valid_neighbours_of_valid_regions** (long width, long height, long *feature_classes, struct **region_descriptor_list** **neighbours)

  Finds every valid neighbour region for every other valid region in a segmentation mask.

- double **kmcc_compute_region_area** (long *feature_classes, long *region_area, long max_classes, long picture_size)

  Computes the area of all regions within feature_ classes.

- long **maximin** (long width, long height, double *L, double *a, double *b, long **motion_info, long cluster_center_position[MAX_MAXIMIN_CLASSES][2], long *feature_classes)

  Implementation of the maximin clustering algorithm.

- void **kmcc_set_classes** (struct **picture_info_struct** *picture_info, long **motion_info, long max_block_displacement, struct **kmcc_centroid** *centroids, long max_classes, long *feature_classes, long width, long height)

  Assign every element of feature_ classes a region descriptor according to it's nearest centroid.

- long **kmcc_reset_centroids_2** (double *L, double *a, double *b, long **motion_info, long max_classes, long width, long height, long *feature_classes, struct **kmcc_centroid** *centroids, long threshold, long index_factor, long index_offset)

  Computes the data for region centoids by averaging the features of every element with the same region descriptor.

- long **kmcc_reset_centroids** (double *L, double *a, double *b, long **motion_info, long max_classes, long width, long height, long *feature_classes, struct **kmcc_centroid** *centroids, long threshold)

  Variant of kmcc_ reset_ centroids_ 2 which considers values of feature_ classes in the range [0:1:max_ classes].

- void **km_clustering** (long width, long height, double *L, double *a, double *b, long **motion_info, long max_block_displacement, struct **kmcc_centroid** *centroids, long number_of_classes, long *feature_classes)

  Performes a k-Means-clustering on given picture.

- long **get_area** (long old_class, long new_class, long x, long y, long *feature_classes, long *new_feature_classes, long width, long height)

  Computes the area of a connected part of a class and copies it to new_feature_classes assigning a new region descriptor new_class.

- double **kmcc_distance_centroids** (struct **kmcc_centroid** *centroids, struct **kmcc_centroid** *new_centroids, long number_of_classes, long old_number_of_classes, long use_spatial_information)

  Computes the average distance between the components of the sets of centroids.

- long **get_region_position** (long *x, long *y, long *feature_classes, long region_descriptor, long width, long height)
- long **merge_with_closest_neighbour_if_below_threshold** (long *new_feature_classes, struct **picture_info_struct** *picture_info, long **motion_info, long use_spatial_information, long number_of_dropped_regions, long number_of_valid_regions, long *help, double threshold)

  Merges all desputed regions (negative index) with neighbouring valid regions, if the minimal centroid distance between the disputed region and the neighbour regions is below a given threshold.

- long **merge_with_closest_neighbour** (long *new_feature_classes, struct **picture_info_struct** *picture_info, long **motion_info, long use_spatial_information, long number_of_dropped_regions, long number_of_valid_regions, long *help)

  Merges all desputed regions (negative index) with neighbouring valid regions (like merge_with_closest_neighbour_if_below_threshold, but performs merging for every disputed region).

- long * **kmcc_evaluate_connectivity** (long *number_of_classes, long *feature_classes, struct **picture_info_struct** *picture_info, long **motion_info, long use_spatial_features, long region_size_threshold, long reassign_dropped_regions)

  Enforces connectivty constraint by relabling non-connected parts of regions and dropping regions which are too small; maintains region descriptors if possible.

- long * **kmcc_clustering** (long width, long height, struct **picture_info_struct** *picture_info, long **motion_info, long max_block_displacement, struct **kmcc_centroid** *centroids, long *number_of_classes, long *feature_classes)

  Implementation of the kmcc clustering algorithm.

- void **average_filter** (double *v, long width, long height)

  Applies a 3x3 moving average filter on a colour component.

- void **mark_disputed_regions** (struct **picture_info_struct** *picture_info, struct **picture_info_struct** *last_picture, long *disputed)

  Markes regions within a given segmentation mask as disputed if the colour difference between two pictures exeeds a threshold.

- void **compute_normalized_histograms_for_objects** (struct **picture_info_struct** *picture_info, long *feature_classes, long number_of_regions, struct **histogram_struct** *histogram)

  Computed the normalized histograms for every region given in a segmantation mask.

- int **mark_region_compare** (const void *e1, const void *e2)

  Compares two values.

- void **mark_region_borders_as_disputed** (long *feature_classes, long width, long height)

  Marks region borders in a given segmention mask as new disputed regions.

- **histogram_struct** * **assign_disputed_pixels_to_neighbour_region** (struct **picture_info_struct** *picture_info, struct **picture_info_struct** *picture_info_for_histograms, long *feature_classes, long *feature_classes_for_histograms, long *number_of_regions, long number_of_disputed_regions, long bool_free_histograms, double *max_probabilities, struct **region_descriptor_list** ***keep_neighbours)

  Assigns pixels within every disputed region to a appropriate neighbouring valid region based on histogram information.

- long **relabel_regions_to_miminmum_numbers** (long *feature_classes, long number_of_regions, struct **picture_info_struct** *picture_info)

  Takes an array of possibly scattered region despriptors and computes the minimal number of regions and relabels the regions appropriately.

- long * **mask_improvement** (long width, long height, struct **picture_info_struct** *picture_info, long **motion_info, long max_block_displacement, long *number_of_regions, long *feature_classes)

  Performs a segmentation mask improvement on a given segmentation mask.

- void **form_output_image_from_mask** (int sub, **file_imagestruct** *output_image, long *pixel_feature_classes, long number_of_regions)

  forms a yuv output image visualizing a given segmentation

- int **write_segmentation_mask_to_disk** (long *segmentation_mask, struct **picture_info_struct** *picture_info)
- void **get_bounding_box_and_size_for_regions** (struct **region_info** *regions, long *feature_classes, long number_of_regions, long frame_count, struct **picture_info_struct** *picture_info)

  Calculates bounding box coordinates for each region; attention: EXTINCT regions get arbitrary values.

- void **update_region_information** (struct **picture_info_struct** *picture_info, struct **region_info** **regions, long number_of_regions, long previous_max_number_of_regions, long *segmentation_mask, long frame_count)

  Updates the information for every region (except motion similarity data which can be updated seperately).

- double * **gaussian_elimination** (long n, double **a)

  Solves an equation system using gaussian elemination.

- **region_motion_parameter** * **get_motion_info_for_regions** (long region_descriptor_1, long region_descriptor_2, long *feature_classes, long **motion_info, struct **picture_info_struct** *picture_info, double *model_parameters)

  Computes motion vector and estimation error for a region or a unity of two regions using a bilinear motion model and least squares estimation or givel model parameters.

- void **update_motion_similarity_between_regions** (long *feature_classes, long number_of_regions, struct **object_info_struct** *obj_info, struct **picture_info_struct** *picture_info, long **motion_info)

  Updates the motion similarity data for every region and their spatiotemporal neighbours resp.

- void **perform_complete_update_on_region_data** (struct **picture_info_struct** *picture_info, struct **object_info_struct** *obj_info, long number_of_regions, long *R)

  Updates the motion similarity data for every region and their spatiotemporal neighbours resp.

- void **detect_new_regions** (long number_of_disputed_regions_R_I, long *R, long *R_E, long *R_I, long number_of_regions_R_I, long *number_of_regions, double *max_probabilities_for_disputed_pixels, struct **histogram_struct** *histograms, struct **region_descriptor_list** **neighbours, struct **picture_info_struct** *picture_info, struct **object_info_struct** *obj_info)

  Perform the detektion of new regions based on an intermediate segmentation containing disputed pixels.

- int **object_segmentation** (int sub, **file_imagestruct** *input_image, **file_imagestruct** *output_image, **filterstruct** *filter)

  function computing an object segmentation on a frame

- int **double_labstar_object_segmentation** (**file_imagestruct** *input_image, **file_imagestruct** *output_image, **filterstruct** *filter)
- void **add_entry_to_merge_list** (long i, long region_to_merge_with, long *merge_list, long *number_of_planned_mergings)

  Adds an entry to a given merge list resolving conflicts with previous mergings.

- **object_info_struct** * **apply_mergings_on_segmentation_and_update_data** (struct **object_info_struct** *obj_info, struct **picture_info_struct** *picture_info, long *merge_list, char write_output_file)

  Applies desired mergings between regions on a whole sequence of segmentation masks and incrementally updates the object information during mergings.

- void **write_video_from_segmentation_file** (struct **object_info_struct** *obj_info, struct **picture_info_struct** *picture_info, int scale_by_max_region_in_mask)

Creates a video visualizing a given sequence of segmentation mask.

- void **free_object_info** (struct **object_info_struct** *obj_info, int free_picture_info)

  This function free the memory pointed to by obj_info and its entries including all concetenated lists.

- double **find_minimal_motion_similarity_distance** (struct **object_info_struct** *obj_info, long *region_descriptor_1, long *region_descriptor_2, long *merge_list)

  Searches the minimal motion similarity distance for all regions.

- int **double_labstar_object_segmentation_exit** (FILE *inoutfile, **filterstruct** *filter)

  Performs mergings after segmentation is finished.

A.1.1 Detailed Description

Performs object segmentation.

Author:
Michael Balda (balda@lnt.de, michael.balda@gmx.de)

Date:
2004

Definition in file **object_segmentation.c**.

A.1.2 Function Documentation

A.1.2.1 void add_entry_to_merge_list (long *i*, long *region_to_merge_with*, long * *merge_list*, long * *number_of_planned_mergings*)

Adds an entry to a given merge list resolving conflicts with previous mergings.

Parameters:

i region which should be merged

region_to_merge_with region which should be merged with i

merge_list list containing all desired mergings so far

number_of_planned_mergings pointer to number of planned mergings so far; is incremented if a new merging is added

Definition at line 4090 of file object_segmentation.c.

References TRUE.

Referenced by double_labstar_object_segmentation_exit().

A.1.2.2 struct object_info_struct* apply_mergings_on_segmentation_and_update_data (struct object_info_struct * *obj_info*, struct picture_info_struct * *picture_info*, long * *merge_list*, char *write_output_file*)

Applies desired mergings between regions on a whole sequence of segmentation masks and incrementally updates the object information during mergings.

Parameters:

obj_info object information for given sequence of segmentation masks

picture_info picture data for some non-empty picute within sequence

merge_list array containing planned mergings

write_output_file switch which toggles writing of output video visualizing segmentation

Returns:

updated object information for sequence with applied mergings

Definition at line 4134 of file object_segmentation.c.

References apply_mergings_on_segmentation_and_update_data(), object_info_struct::blockdim, object_info_struct::existing_and_extinct_region_info, object_info_struct::frame_count, MALLOC, perform_complete_update_on_region_data(), object_info_struct::picture_info, object_info_struct::previous_max_number_of_regions, and region_info::status.

Referenced by apply_mergings_on_segmentation_and_update_data(), and double_labstar_object_segmentation_exit().

A.1.2.3 struct histogram_struct* assign_disputed_pixels_to_neighbour_region (struct picture_info_struct * *picture_info*, struct picture_info_struct * *picture_info_for_histograms*, long * *feature_classes*, long * *feature_classes_for_histograms*, long * *number_of_regions*, long *number_of_disputed_regions*, long *bool_free_histograms*, double * *max_probabilities*, struct region_descriptor_list *** *keep_neighbours*)

Assigns pixels within every disputed region to a appropriate neighbouring valid region based on histogram information.

Parameters:

picture_info information on the picture

picture_info_for_histograms picture info which is used for the computation of the valid regions histograms

feature_classes segmentation mask containing valid and disputed regions

feature_classes_for_histograms segmentation mask containing the position of the valid regions on which the histograms are to be computed

number_of_regions maximal number of valid regions in both segmentation masks

number_of_disputed_regions number of disputed region in feature_classes

bool_free_histograms switch indication whether the histogram data should be freed or kept

max_probabilities double array of same size as feature_classes where the maximal probablilities for every disputed pixel is stored; if NULL these probabilities are not stored at all

keep_neighbours pointer to pointer to array where neighbourhood-infomation is stored; if NULL neighbourhood pointer won't be set and the neighbourhood information is dropped

Returns:

pointer to array of histogram info for every region or NULL if bool_free_-histogramms equals TRUE

Definition at line 1853 of file object_segmentation.c.

References assign_disputed_pixels_to_neighbour_region(), compute_-normalized_histograms_for_objects(), get_valid_neighbours_of_disputed_-regions(), region_descriptor_list::next, region_descriptor_list_delete_first(), TRUE, and region_descriptor_list::value.

Referenced by assign_disputed_pixels_to_neighbour_region(), and object_-segmentation().

A.1.2.4 void average_filter (double * *v*, long *width*, long *height*)

Applies a 3x3 moving average filter on a colour component.

Parameters:

v array containing colour information

width horizontal dimension of the image

height vertical dimension of the image

Definition at line 1634 of file object_segmentation.c.

Referenced by object_segmentation().

A.1.2.5 void compute_normalized_histograms_for_objects (struct picture_info_struct * *picture_info*, long * *feature_classes*, long *number_of_regions*, struct histogram_struct * *histogram*)

Computed the normalized histograms for every region given in a segmantation mask.

Parameters:

picture_info infomation on first picture

feature_classes segmentation mask

number_of_regions maximal number of regions appearing in feature_-classes

histogram array where the histogram information is copied to

Definition at line 1687 of file object_segmentation.c.

References picture_info_struct::a, picture_info_struct::b, HISTOGRAM_ENTRIES, picture_info_struct::L, picture_info_struct::max_a, picture_info_struct::max_b, picture_info_struct::max_L, picture_info_struct::min_a, picture_info_struct::min_b, picture_info_struct::min_L, and picture_info_struct::picture_size.

Referenced by assign_disputed_pixels_to_neighbour_region().

A.1.2.6 int copy_file (char * *src_filename*, char * *dst_filename*)

This function copies a file on the disc.

Parameters:

src_filename The filename of the source

dst_filename The filename where the source is written to

Returns:

RET_OK in case of success

Definition at line 363 of file object_segmentation.c.

References DBG_ERROR, dbg_out(), RET_FATAL, and RET_OK.

A.1.2.7 void detect_new_regions (long *number_of_disputed_regions_R_I*, long * *R*, long * *R_E*, long * *R_I*, long *number_of_regions_R_I*, long * *number_of_regions*, double * *max_probabilities_for_disputed_pixels*, struct histogram_struct * *histograms*, struct region_descriptor_list ** *neighbours*, struct picture_info_struct * *picture_info*, struct object_info_struct * *obj_info*)

Perform the detektion of new regions based on an intermediate segmentation containing disputed pixels.

Parameters:

number_of_disputed_regions_R_I number of disputed regions in mask R_I

R segmentation mask where the final semgentation is copied to

R_E segmentation mask containing the motion traced regions

R_I intermediate segmentation mask containing disputed pixels due to changes between two frames

number_of_regions_R_I number of valid regions in mask R_I

number_of_regions number of regions in mask R_E

max_probabilities_for_disputed_pixels array containing the maximal probabilities for the disuted pixels (by-product from generation of mask R_E)

histograms array of histogram structs for every valid region in R_I

neighbours array of neighbour-lists for every disputed region

picture_info picture data

obj_info information on every region in R_E

Definition at line 3096 of file object_segmentation.c.

References kmcc_centroid::a, picture_info_struct::a, kmcc_centroid::b, picture_info_struct::b, histogram_struct::bin_size_a, histogram_struct::bin_size_b, histogram_struct::bin_size_L, C_TH, COLOR_DISTANCE_THRESHOLD_SEGMENTATION_MASK_FUSION_C_TH, object_info_struct::existing_and_extinct_region_info, picture_info_struct::height, histogram_struct::histogram_a, histogram_struct::histogram_b, HISTOGRAM_ENTRIES, histogram_struct::histogram_L, kmcc_evaluate_connectivity(), kmcc_reset_centroids(), kmcc_centroid::L, picture_info_struct::L, MALLOC, merge_with_closest_neighbour_if_below_threshold(), picture_info_struct::min_a, picture_info_struct::min_b, picture_info_struct::min_L, region_descriptor_list::next, picture_info_struct::picture_size, object_info_struct::previous_max_number_of_regions, region_info::region_centroid, region_descriptor_list_delete_first(), sideinfo_add_with_identifier(), sqr_object(), region_info::status, TRUE, region_descriptor_list::value, picture_info_struct::width, kmcc_centroid::x, and kmcc_centroid::y.

Referenced by object_segmentation().

A.1.2.8 double distance_kmcc (double * *pix_color*, double * *region_color*, double * *pix_motion*, double * *region_motion*, long *pix_x*, long *pix_y*, long *region_x*, long *region_y*, long *region_area*, double *average_region_area*, double *DI_max*, long *max_block_displacement_x*, long *max_block_displacement_y*, long *width*, long *height*)

Computes the distance between a feature vector and a region center w.r.t.

color, motion and picture coordinates.

Parameters:

pix_color array of the 3 la*b* color componentes of the pixel

region_color array of the 3 la*b* color componentes assigned to the region

pix_motion array of the 2 motion componentes of the pixel

region_motion array of the 2 motion componentes assigned to the region

pix_x x coordinate of the pixel

pix_y y coordinate of the pixel

region_x x coordinate of the region center

region_y y coordinate of the region center

region_area area of the region

average_region_area average area of all regions

DI_max estimation of image contrast

max_block_displacement_x maximal block displacement of block motion estimation in x direction

max_block_displacement_y maximal block displacement of block motion estimation in y direction

width number of elements in x-direction

height number of elements in y-direction

Returns:
distance between feature vector and region center

Definition at line 300 of file object_segmentation.c.

References euclid_distance(), and sqr_object().

Referenced by kmcc_set_classes().

A.1.2.9 double distance_maximin (double * *pix_color*, double * *region_color*, double * *pix_motion*, double * *region_motion*, long *pix_x*, long *pix_y*, long *region_x*, long *region_y*)

Computes the distance between a feature vector and a region center w.r.t.

color, motion and picture coordinates.

Parameters:
pix_color array of the 3 la*b* color componentes of the pixel

region_color array of the 3 la*b* color componentes assigned to the region

pix_motion array of the 2 motion componentes of the pixel

region_motion array of the 2 motion componentes assigned to the region

pix_x x coordinate of the pixel

pix_y y coordinate of the pixel

region_x x coordinate of the region center

region_y y coordinate of the region center

Returns:
distance between feature vector and region center

Definition at line 259 of file object_segmentation.c.

References euclid_distance().

Referenced by maximin().

A.1.2.10 int double_labstar_object_segmentation_exit (FILE * *inoutfile*, filterstruct * *filter*)

Performs mergings after segmentation is finished.

Parameters:
inoutfile isn't used contains all information necessacry for merging

Returns:
RET_OK if everything is okay

Definition at line 4323 of file object_segmentation.c.

References add_entry_to_merge_list(), apply_mergings_on_segmentation_and_update_data(), DBG_ERROR, dbg_out(), object_info_struct::existing_and_extinct_region_info, find_minimal_motion_similarity_distance(), object_info_struct::frame_count, free_object_info(), MALLOC, region_descriptor_list::next, object_info_struct::picture_info, region_info::presence_monitor, object_info_struct::previous_max_number_of_regions, REGION_MERGING_MOTION_SIMILARITY_THRESHOLD, region_info::region_size, RET_FATAL, RET_OK, region_info::spatiotemporal_neighbours, sqr_object(), TRUE, region_descriptor_list::value, region_descriptor_list::value2, region_descriptor_list::value3, and write_video_from_segmentation_file().

A.1.2.11 double estimate_DImax (struct picture_info_struct * *picture_info*)

Computes an estimation of the image contrast; necessary for distance measure.

Parameters:

picture_info contains all the picture information needed

Returns:

estimated image contrast

Definition at line 398 of file object_segmentation.c.

References picture_info_struct::a, picture_info_struct::b, picture_info_struct::DI_max, picture_info_struct::L, picture_info_struct::max_a, picture_info_struct::max_b, picture_info_struct::max_L, picture_info_struct::min_a, picture_info_struct::min_b, picture_info_struct::min_L, picture_info_struct::picture_size, and sqr_object().

Referenced by km_clustering(), and object_segmentation().

A.1.2.12 double euclid_distance (double * *vec1*, double * *vec2*, long *dim*)

Computes the euclid distance between two vectors.

Parameters:

vec array of the componenents of 1st vector

vec2 array of the componenents of 2nd vector

dim number of elements of one vector

Returns:

euclid distance between vectors

Definition at line 147 of file object_segmentation.c.

References sqr_object().

Referenced by distance_kmcc(), and distance_maximin().

A.1.2.13 double find_minimal_motion_similarity_distance (struct object_info_struct * *obj_info*, long * *region_descriptor_1*, long * *region_descriptor_2*, long * *merge_list*)

Searches the minimal motion similarity distance for all regions.

Parameters:

obj_info information on current segmentation

region_descriptor_1 descriptor of region with min motion distance will be copied here

region_descriptor_2 descriptor of other region with min motion distance will be copied here

merge_list list of all conducted merges

Returns:

minimum motion similatity distance

Definition at line 4288 of file object_segmentation.c.

References object_info_struct::existing_and_extinct_region_info, region_descriptor_list::next, object_info_struct::previous_max_number_of_regions, region_info::spatiotemporal_neighbours, region_descriptor_list::value, region_descriptor_list::value2, and region_descriptor_list::value3.

Referenced by double_labstar_object_segmentation_exit().

A.1.2.14 void form_output_image_from_mask (int *sub*, file_imagestruct * *output_image*, long * *pixel_feature_classes*, long *number_of_regions*)

forms a yuv output image visualizing a given segmentation

Parameters:

sub value indicating the colour-subsampling rate for yuv-image

output_image pointer struct to which the yuv-image is copied to

pixel_feature_classes segmentation mask

number_of_regions maximal number of regions in segmentation mask

Definition at line 2233 of file object_segmentation.c.

References int_file_imagestruct::height, MALLOC, SUBX, int_file_imagestruct::U, int_file_imagestruct::V, int_file_imagestruct::width, and int_file_imagestruct::Y.

Referenced by object_segmentation().

A.1.2.15 void free_object_info (struct object_info_struct * *obj_info*, int *free_picture_info*)

This function free the memory pointed to by obj_info and its entries including all concetenated lists.

Parameters:

obj_info pointer object info struct which should be freed

free_picture_info if 0 picture info struct will not be freed

Definition at line 4251 of file object_segmentation.c.

References picture_info_struct::a, picture_info_struct::b, object_info_struct::existing_and_extinct_region_info, picture_info_struct::L, object_info_struct::picture_info, object_info_struct::previous_mask, object_info_struct::previous_max_number_of_regions, and region_descriptor_list_delete_first().

Referenced by double_labstar_object_segmentation_exit().

A.1.2.16 double* gaussian_elimination (long *n*, double ** *a*)

Solves an equation system using gaussian elemination.

Parameters:

n number of equations and unknowns resp.

a augumented matrix ([n][n+1] lines indicated by first index) consisting of n lines and n+1 colums; last column is RHS vector

Returns:

pointer to solution (double-array of lenght n); NULL if no unique solution exists

Definition at line 2482 of file object_segmentation.c.

References MALLOC, and TRUE.

Referenced by get_motion_info_for_regions().

A.1.2.17 long get_area (long *old_class*, long *new_class*, long *x*, long *y*, long * *feature_classes*, long * *new_feature_classes*, long *width*, long *height*)

Computes the area of a connected part of a class and copies it to new_feature_classes assigning a new region descriptor new_class.

Parameters:

old_class number of the region for which the area is computed

new_class descriptor which is used for the copy of the connected part of the region in new_feature_classes

x seed position from which the connectivity of the unterlying region is evaluted

y seed position from which the connectivity of the unterlying region is evaluted

feature_classes array containing the region descriptors of each picture element

new_feature_classes array to which the connected part of the region is copied

width number of picture elements in x direction

height number of picture elements in y direction

Returns:
area of the region

Definition at line 1088 of file object_segmentation.c.

References CALLOC, and TRUE.

A.1.2.18 void get_bounding_box_and_size_for_regions (struct region_info * *regions*, long * *feature_classes*, long *number_of_regions*, long *frame_count*, struct picture_info_struct * *picture_info*)

Calculates bounding box coordinates for each region; attention: EXTINCT regions get arbitrary values.

Parameters:
regions array of **region_info** structs where bounding box coordinates are written to

feature_classes region descriptor data

number_of_regions the highest region descriptor in feature_classes + 1

picture_info information about picture dimensions

Definition at line 2289 of file object_segmentation.c.

References region_info::bounding_box, picture_info_struct::height, MALLOC, region_info::region_size, and picture_info_struct::width.

Referenced by object_segmentation(), and perform_complete_update_on_region_data().

A.1.2.19 struct region_motion_parameter* get_motion_info_for_regions (long *region_descriptor_1*, long *region_descriptor_2*, long * *feature_classes*, long ** *motion_info*, struct picture_info_struct * *picture_info*, double * *model_parameters*)

Computes motion vector and estimation error for a region or a unity of two regions using a bilinear motion model and least squares estimation or givel model parameters.

Parameters:
region_descriptor_1 region descriptor of first region in feature_classes; may include non-existing region descriptors

region_descriptor_2 region descriptor of second region in feature_classes; same as one if E_avg should be computed for one region only

feature_classes array containing region information

blockdim dimension of blocks from motion estimation

picture_info information about picture dimensions

model_parameters these are the parameters which are to be used for region motion computation; if NULL optimized model parameters are computed and used

Returns:
region_motion_vector struct containing motion data for region

Definition at line 2601 of file object_segmentation.c.

References region_motion_parameter::a, region_motion_parameter::E_avg, gaussian_elimination(), get_motion_info_for_regions(), MALLOC, region_motion_parameter::number_of_nonrejected_elements, and sqr_object().

Referenced by get_motion_info_for_regions(), and update_motion_similarity_between_regions().

A.1.2.20 void get_pixel_neighbour_regions (long *pixel_neighbour_regions*[8], long *x*, long *y*, long *width*, long * *feature_classes*)

Finds any region descriptor of the 8 neighbouring pixels.

Parameters:
pixel_neighbour_regions array to which the neighbouring region descriptors are copied to

x horizontal location of considered pixel

y vertical location of considered pixel

width width of the segmentation mask

feature_classes array containing the segmentation mask

Definition at line 534 of file object_segmentation.c.

Referenced by get_valid_neighbours_of_disputed_regions(), and get_valid_neighbours_of_valid_regions().

A.1.2.21 void get_valid_neighbours_of_disputed_regions (long *width*, long *height*, long * *feature_classes*, struct region_descriptor_list ** *neighbours*)

Finds every valid neighbour region for every disputed region in a segmentation mask.

Parameters:
width width of the segmentation mask

height height of the segmentation mask

feature_classes array containing segmentation mask

neighbours array of pointers to region descriptor lists to which the neighbour information is copied to

Definition at line 553 of file object_segmentation.c.

References get_pixel_neighbour_regions(), and region_descriptor_list_append().

Referenced by assign_disputed_pixels_to_neighbour_region(), mask_improvement(), and merge_with_closest_neighbour_if_below_threshold().

A.1.2.22 void get_valid_neighbours_of_valid_regions (long *width*, long *height*, long * *feature_classes*, struct region_descriptor_list ** *neighbours*)

Finds every valid neighbour region for every other valid region in a segmentation mask.

Parameters:

width width of the segmentation mask

height height of the segmentation mask

feature_classes array containing segmentation mask

neighbours array of pointers to region descriptor lists to which the neighbour information is copied to

Definition at line 611 of file object_segmentation.c.

References get_pixel_neighbour_regions(), and region_descriptor_list_append().

Referenced by update_region_information().

A.1.2.23 void km_clustering (long *width*, long *height*, double * *L*, double * *a*, double * *b*, long ** *motion_info*, long *max_block_displacement*, struct kmcc_centroid * *centroids*, long *number_of_classes*, long * *feature_classes*)

Performes a k-Means-clustering on given picture.

Parameters:

width number of elements in x-direction

height number of elements in y-direction

L Luminance of picture elements

a Chrominance of picture elements

b Chrominance of picture elements

motion_info motion features for every picture_element

max_block_displacement maximal length of a motion vector from motion info

centroids array for the centroid information, expected to be filled with initial values

number_of_classes number of centroids

feature_classes array for the final region assigned to each picture element

Definition at line 1058 of file object_segmentation.c.

References picture_info_struct::a, picture_info_struct::b, estimate_DImax(), picture_info_struct::height, KM_THRESHOLD, kmcc_reset_centroids(), kmcc_set_classes(), picture_info_struct::L, picture_info_struct::picture_size, and picture_info_struct::width.

Referenced by object_segmentation().

A.1.2.24 long* kmcc_clustering (long *width*, long *height*, struct picture_info_struct * *picture_info*, long ** *motion_info*, long *max_block_displacement*, struct kmcc_centroid * *centroids*, long * *number_of_classes*, long * *feature_classes*)

Implementation of the kmcc clustering algorithm.

Parameters:

picture_info **picture_info_struct** containing all necessary information about the picture

motion 2D-array of motion vectors [x or y component][number of feature]

max_block_displacement maximal block displacement of block motion estimation

centroids initial centroids as input and final centroids as output

number_of_classes number of classes for KMCC

Returns:

pointer to array with segmentation mask

Definition at line 1534 of file object_segmentation.c.

Referenced by object_segmentation().

A.1.2.25 double kmcc_compute_region_area (long * *feature_classes*, long * *region_area*, long *max_classes*, long *picture_size*)

Computes the area of all regions within feature_classes.

Parameters:

feature_classes contains the information about the regions

region_area previously allocated array for the region's areas

max_classes maximal number of classes within feature_classes

picture_size size of the picture in pixels

Returns:

average region size

Definition at line 642 of file object_segmentation.c.

Referenced by kmcc_set_classes(), and mask_improvement().

A.1.2.26 double kmcc_distance_centroids (struct kmcc_centroid * *centroids*, struct kmcc_centroid * *new_centroids*, long *number_of_classes*, long *old_number_of_classes*, long *use_spatial_information*)

Computes the average distance between the components of the sets of centroids.

Parameters:

centroids first set of centroids

new_centroids second set of centroids

number_of_classes number of centroids in second set

old_number_of_classes number of centroids in first set

use_spatial_infomation toggles whether spatial centroid infomation should be used

Returns:
average centroid distance

Definition at line 1204 of file object_segmentation.c.

References sqr_object().

Referenced by merge_with_closest_neighbour_if_below_threshold().

A.1.2.27 long* kmcc_evaluate_connectivity (long * *number_of_classes*, long * *feature_classes*, struct picture_info_struct * *picture_info*, long ** *motion_info*, long *use_spatial_features*, long *region_size_threshold*, long *reassign_dropped_regions*)

Enforces connectivty constraint by relabling non-connected parts of regions and dropping regions which are too small; maintains region descriptors if possible.

Parameters:

number_of_classes number of regions present in feature_classes; may be changed by this function; if 0 is passed a complete relabeling is performed and the true number of regions is evaluated

feature_classes array containing the region infomation

picture_info struct containing picture infomation

motion_info motion infoformation which is used for merging small regions; pass NULL if motion should not be considered

use_spatial_features pass TRUE if spatial features should be used for the decision to which region too small regions should be added

region_size_threshold minimal size for regions; if a region is smaller it's marked as disputed and may be added to a neighbour region if reassign_dropped_regions is set to TRUE

reassign_dropped_regions TRUE if small regions are to be reassigned to neighbours; if FALSE these regions are just marked as disputed

Returns:
TRUE if every region is assigned correctly, FALSE may indicate that a further iteration is necessary (e.g. if no direct neighbour is found for a disputed region)

Definition at line 1399 of file object_segmentation.c.

Referenced by detect_new_regions(), and object_segmentation().

A.1.2.28 long kmcc_reset_centroids (double * *L*, double * *a*, double * *b*, long ** *motion_info*, long *max_classes*, long *width*, long *height*, long * *feature_classes*, struct kmcc_centroid * *centroids*, long *threshold*)

Variant of kmcc_reset_centroids_2 which considers values of feature_classes in the range [0:1:max_classes].

Parameters:

L Luminance of picture elements

a Chrominance of picture elements

b Chrominance of picture elements

motion_info motion features for every picture_element

max_classes maximial number of classes in feature_classes

width number of elements in x-direction

height number of elements in y-direction

array containing the region descriptors for picture elements

centroids array for the centroid information

threshold a distance threshold for every centroid-data element to determine how many centroids changed their position in comparison to their previous values

Returns:

number of centroids which have changed their position

Definition at line 1041 of file object_segmentation.c.

References kmcc_reset_centroids_2().

Referenced by detect_new_regions(), km_clustering(), mask_improvement(), merge_with_closest_neighbour_if_below_threshold(), and update_region_information().

A.1.2.29 long kmcc_reset_centroids_2 (double * *L*, double * *a*, double * *b*, long ** *motion_info*, long *max_classes*, long *width*, long *height*, long * *feature_classes*, struct kmcc_centroid * *centroids*, long *threshold*, long *index_factor*, long *index_offset*)

Computes the data for region centoids by averaging the features of every element with the same region descriptor.

Parameters:

L Luminance of picture elements

a Chrominance of picture elements

b Chrominance of picture elements

motion_info motion features for every picture_element

max_classes maximial number of classes in feature_classes

width number of elements in x-direction

height number of elements in y-direction

array containing the segmentation mask

centroids array for the centroid information

threshold a distance threshold for every centroid-data element to determine how many centroids changed their position in comparison to their previous values

index_factor a factor to scale the elements of feature_classes, so that the elements match into the range [0:1:max_classes] whereas negative values will be ignored

index_offset an offset, which will be added to the values in feature classes so that they match the condition mentioned above

Returns:

number of centroids which have changed their position

Definition at line 924 of file object_segmentation.c.

References kmcc_centroid::a, kmcc_centroid::b, kmcc_centroid::L, kmcc_centroid::motion_x, kmcc_centroid::motion_y, kmcc_centroid::x, and kmcc_centroid::y.

Referenced by kmcc_reset_centroids(), and merge_with_closest_neighbour_if_below_threshold().

A.1.2.30 void kmcc_set_classes (struct picture_info_struct * *picture_info*, long ** *motion_info*, long *max_block_displacement*, struct kmcc_centroid * *centroids*, long *max_classes*, long * *feature_classes*, long *width*, long *height*)

Assign every element of feature_classes a region descriptor according to it's nearest centroid.

Parameters:

picture_info information on the picture

motion_info information on motion information

centroids array containing the data of every centroid

max_classes maximum number of classes

feature_classes array which will be filled with the region descriptors

width number of elements in x-direction

height number of elements in y-direction

Definition at line 798 of file object_segmentation.c.

References picture_info_struct::a, picture_info_struct::b, picture_info_struct::DI_max, distance_kmcc(), kmcc_compute_region_area(), kmcc_centroid::L, picture_info_struct::L, MALLOC, and TRUE.

Referenced by km_clustering().

A.1.2.31 void mark_disputed_regions (struct picture_info_struct * *picture_info*, struct picture_info_struct * *last_picture*, long * *disputed*)

Markes regions within a given segmentation mask as disputed if the colour difference between two pictures exeeds a threshold.

Parameters:

picture_info infomation on first picture

last_picture infomation on other picture

mask in which the pixels should be masked as disputed if necessary

Definition at line 1661 of file object_segmentation.c.

References picture_info_struct::a, picture_info_struct::b, DIF_TH, picture_info_struct::L, picture_info_struct::picture_size, and sqr_object().

Referenced by object_segmentation().

A.1.2.32 void mark_region_borders_as_disputed (long * *feature_classes*, long *width*, long *height*)

Marks region borders in a given segmention mask as new disputed regions.

Parameters:

feature_classes segmentation mask in which the borders are marked as disputed regions

width width of the segmentation mask

height height of the segmentation mask

Definition at line 1808 of file object_segmentation.c.

References MALLOC, and mark_region_compare().

A.1.2.33 int mark_region_compare (const void * *e1*, const void * *e2*)

Compares two values.

Parameters:

pointer to first value

pointer to second value

Returns:

-1 if first value is greater, 1 for second and 0 if equal

Definition at line 1794 of file object_segmentation.c.

Referenced by mark_region_borders_as_disputed().

A.1.2.34 long* mask_improvement (long *width*, long *height*, struct picture_info_struct * *picture_info*, long ** *motion_info*, long *max_block_displacement*, long * *number_of_regions*, long * *feature_classes*)

Performs a segmentation mask improvement on a given segmentation mask.

Parameters:

width width of picture and semgentation mask

height height of picture and semgentation mask

picture_info picture data array containing motion vectors maximal block displacement of motion estimation pointer to value of maximal number of regions, may be changed segmentation mask which is to be improved

Returns:

pointer to improved segmentation mask

Definition at line 2035 of file object_segmentation.c.

References picture_info_struct::a, picture_info_struct::b, picture_info_struct::DI_max, get_valid_neighbours_of_disputed_regions(), kmcc_compute_region_area(), kmcc_reset_centroids(), picture_info_struct::L, MALLOC, MASK_IMPROVEMENT_COLOR_DIST_THRESHOLD, region_descriptor_list::next, region_descriptor_list_append(), region_descriptor_list_delete_first(), sqr_object(), and region_descriptor_list::value.

Referenced by object_segmentation().

A.1.2.35 long maximin (long *width*, long *height*, double * *L*, double * *a*, double * *b*, long ** *motion_info*, long *cluster_center_position*[MAX_MAXIMIN_CLASSES][2], long * *feature_classes*)

Implementation of the maximin clustering algorithm.

Parameters:

width number of elements in x-direction

height number of elements in y-direction

L array of LA*B* luminance values of a pixel or block

a array of LA*B* a chrominance values of a pixel or block

b array of LA*B* b chrominance values of a pixel or block

motion_info 2D-array of motion vectors [x or y component][number of feature]

cluster_center_postion [MAX_MAXIMIN_CLASSES][2]-array for the position of cluster_centers

feature_classes array of length width*height containing the number of the class for every block/pixel

Returns:

number of clusters

Definition at line 679 of file object_segmentation.c.

References distance_maximin(), and MAX_MAXIMIN_CLASSES.

Referenced by object_segmentation().

A.1.2.36 long merge_with_closest_neighbour (long * *new_feature_classes*, struct picture_info_struct * *picture_info*, long ** *motion_info*, long *use_spatial_information*, long *number_of_dropped_regions*, long *number_of_valid_regions*, long * *help*)

Merges all desputed regions (negative index) with neighbouring valid regions (like merge_with_closest_neighbour_if_below_threshold, but performs merging for every disputed region).

Parameters:

new_feature_classes array containing the region information

picture_info struct containing the picture information

motion_info motion vectors, may be NULL if motion information should not be considered

use_spatial_information pass TRUE if spatial features should be used to compute the centroid distance

number_of_dropped_regions number of regions that should be reassigned (must not be smaller than the actual number, may be larger)

number_of_valid_regions number of valid regions (positiv index starting from 0)

help array of same size as new_feature_classes contains 1 at points belonging to disputed pixel, 0 for valid region pixels; may be NULL

Returns:

TRUE if every region is assigned correctly, FALSE may indicate that a further iteration is necessary (e.g. if no direct neighbour is found for a disputed region)

Definition at line 1384 of file object_segmentation.c.

A.1.2.37 long merge_with_closest_neighbour_if_below_threshold (long * *new_feature_classes*, struct picture_info_struct * *picture_info*, long ** *motion_info*, long *use_spatial_information*, long *number_of_dropped_regions*, long *number_of_valid_regions*, long * *help*, double *threshold*)

Merges all desputed regions (negative index) with neighbouring valid regions, if the minimal centroid distance between the disputed region and the neighbour regions is below a given threshold.

Parameters:

new_feature_classes array containing the region information

picture_info struct containing the picture information

motion_info motion vectors, may be NULL if motion information should not be considered

use_spatial_information pass TRUE if spatial features should be used to compute the centroid distance

number_of_dropped_regions number of regions that should be reassigned (must not be smaller than the actual number, may be larger)

number_of_valid_regions number of valid regions (positiv index starting from 0)

help array of same size as new_feature_classes contains 1 at points belonging to disputed pixel, 0 for valid region pixels; may be NULL

threshold the distance threshold which has to be underrun so that merging is performed

Returns:

TRUE if every region is assigned correctly, FALSE may indicate that a further iteration is necessary (e.g. if no direct neighbour is found for a disputed region)

Definition at line 1250 of file object_segmentation.c.

References picture_info_struct::a, picture_info_struct::b, get_valid_neighbours_of_disputed_regions(), picture_info_struct::height, kmcc_distance_centroids(), kmcc_reset_centroids(), kmcc_reset_centroids_2(), picture_info_struct::L, MALLOC, picture_info_struct::picture_size, region_descriptor_list_delete_first(), TRUE, and picture_info_struct::width.

Referenced by detect_new_regions().

A.1.2.38 int object_segmentation (int *sub*, file_imagestruct * *input_image*, file_imagestruct * *output_image*, filterstruct * *filter*)

function computing an object segmentation on a frame

Parameters:

sub colour subsampling information

input_image image for which segmentation should be generated

output_image visualisition of current segmentation

filter information for previous run on previous frame

Returns:

RET_OK if no errors occured

Definition at line 3510 of file object_segmentation.c.

References picture_info_struct::a, assign_disputed_pixels_to_neighbour_region(), average_filter(), picture_info_struct::b, object_info_struct::blockdim, int_file_imagestruct::Da, int_file_imagestruct::Db, detect_new_regions(), int_file_imagestruct::DL, estimate_DImax(), object_info_struct::existing_and_extinct_region_info, form_output_image_from_mask(), object_info_struct::frame_count, get_bounding_box_and_size_for_regions(), picture_info_struct::height, int_file_imagestruct::height, km_clustering(), kmcc_clustering(), kmcc_evaluate_connectivity(), kmcc_centroid::L, picture_info_struct::L, MALLOC, mark_disputed_regions(), mask_improvement(),

MAX_MAXIMIN_CLASSES, maximin(), perform_complete_update_on_region_data(), object_info_struct::picture_info, picture_info_struct::picture_size, object_info_struct::previous_mask, object_info_struct::previous_max_number_of_regions, RET_FATAL, RET_OK, sideinfo_add_with_identifier(), TRUE, update_motion_similarity_between_regions(), update_region_information(), picture_info_struct::width, int_file_imagestruct::width, kmcc_centroid::x, and kmcc_centroid::y.

A.1.2.39 void perform_complete_update_on_region_data (struct picture_info_struct * *picture_info*, struct object_info_struct * *obj_info*, long *number_of_regions*, long * *R*)

Updates the motion similarity data for every region and their spatiotemporal neighbours resp.

Parameters:

feature_classes current segmentation mask

number_of_regions maximal number of regions in segmentation mask

obj_info current object information for which region motion info is updated

picture_info picture data

motion_info motion vectors for whole picture

Definition at line 3022 of file object_segmentation.c.

References object_info_struct::blockdim, DBG_ERROR, dbg_out(), object_info_struct::existing_and_extinct_region_info, object_info_struct::frame_count, get_bounding_box_and_size_for_regions(), picture_info_struct::height, MALLOC, picture_info_struct::picture_size, object_info_struct::previous_max_number_of_regions, update_motion_similarity_between_regions(), update_region_information(), and picture_info_struct::width.

Referenced by apply_mergings_on_segmentation_and_update_data(), and object_segmentation().

A.1.2.40 struct region_descriptor_list* region_descriptor_list_append (struct region_descriptor_list * *list_head*, long *value*, long *list_id*)

Appends an element at the end of the list if not already present in list.

Parameters:

list_head pointer to list head

value value for new element

list_id list may be associated with a list id for debugging

Returns:

pointer to list head; may differ from parameter list_head if an empty list was passed

Definition at line 175 of file object_segmentation.c.

References MALLOC, region_descriptor_list::next, region_descriptor_list_append(), region_descriptor_list::value, region_descriptor_list::value2, and region_descriptor_list::value3.

Referenced by get_valid_neighbours_of_disputed_regions(), get_valid_neighbours_of_valid_regions(), mask_improvement(), region_descriptor_list_append(), and update_region_information().

A.1.2.41 struct region_descriptor_list* region_descriptor_list_delete_first (struct region_descriptor_list * *list*)

Deletes first element of list.

Parameters:

list pointer to list head

Returns:

pointer to new list head; may be NULL if list is empty

Definition at line 231 of file object_segmentation.c.

References region_descriptor_list::next, and region_descriptor_list_delete_first().

Referenced by assign_disputed_pixels_to_neighbour_region(), detect_new_regions(), free_object_info(), mask_improvement(), merge_with_closest_neighbour_if_below_threshold(), region_descriptor_list_delete_first(), and update_region_information().

A.1.2.42 void region_descriptor_list_init (struct region_descriptor_list * *list*) [inline]

Initializes a region descriptor list.

Parameters:

list pointer to head of a list

Definition at line 163 of file object_segmentation.c.

A.1.2.43 int region_descriptor_list_is_in (struct region_descriptor_list * *list*, long *value*)

Returns TRUE if element "value" is in list, FALSE elsewise.

Parameters:

list pointer to list head

value value for element to find

Returns:

TRUE or FALSE according to existance of element value in list

Definition at line 215 of file object_segmentation.c.

References region_descriptor_list::next, TRUE, and region_descriptor_list::value.

Referenced by update_region_information().

A.1.2.44 long relabel_regions_to_miminmum_numbers (long * *feature_classes*, long *number_of_regions*, struct picture_info_struct * *picture_info*)

Takes an array of possibly scattered region despriptors and computes the minimal number of regions and relabels the regions appropriately.

Parameters:

feature_classes array containing region descriptors for every picture element

number_of_regions number of regions used in feature classes

picture_info information on the picture

Returns:

new number of classes used in feature_classes

Definition at line 2001 of file object_segmentation.c.

References MALLOC, and picture_info_struct::picture_size.

A.1.2.45 double sqr_object (double *x*) [inline]

Computes the square of a floating point number.

Parameters:

x number to be squared

Returns:

square of x

Definition at line 136 of file object_segmentation.c.

Referenced by detect_new_regions(), distance_kmcc(), double_labstar_object_segmentation_exit(), estimate_DImax(), euclid_distance(), get_motion_info_for_regions(), kmcc_distance_centroids(), mark_disputed_regions(), and mask_improvement().

A.1.2.46 void update_motion_similarity_between_regions (long * *feature_classes*, long *number_of_regions*, struct object_info_struct * *obj_info*, struct picture_info_struct * *picture_info*, long ** *motion_info*)

Updates the motion similarity data for every region and their spatiotemporal neighbours resp.

Parameters:

feature_classes current segmentation mask

number_of_regions maximal number of regions in segmentation mask

obj_info current object information for which region motion info is updated

picture_info picture data

motion_info motion vectors for whole picture on pixel basis

Definition at line 2907 of file object_segmentation.c.

References region_motion_parameter::a, region_motion_parameter::E_avg, object_info_struct::existing_and_extinct_region_info, FP_INFINITE, get_motion_info_for_regions(), region_descriptor_list::next, region_descriptor_list::value, region_descriptor_list::value2, and region_descriptor_list::value3.

Referenced by object_segmentation(), and perform_complete_update_on_region_data().

A.1.2.47 void update_region_information (struct picture_info_struct * *picture_info*, struct region_info ** *regions*, long *number_of_regions*, long *previous_max_number_of_regions*, long * *segmentation_mask*, long *frame_count*)

Updates the information for every region (except motion similarity data which can be updated seperately).

Parameters:

picture_info picture data which is used to compute several indicators

regions pointer to array containg region information, may be changed

number_of_regions new number of regions in segmentation mask

previous_max_number_of_regions number of regions when last update was performed

segmentation_mask current segmentation mask

frame_count current number of frame

Definition at line 2341 of file object_segmentation.c.

References picture_info_struct::a, picture_info_struct::b, get_valid_neighbours_of_valid_regions(), picture_info_struct::height, kmcc_reset_centroids(), picture_info_struct::L, MALLOC, region_descriptor_list_append(), region_descriptor_list_delete_first(), region_descriptor_list_is_in(), TRUE, region_descriptor_list::value, and picture_info_struct::width.

Referenced by object_segmentation(), and perform_complete_update_on_region_data().

A.1.2.48 void write_video_from_segmentation_file (struct object_info_struct * *obj_info*, struct picture_info_struct * *picture_info*, int *scale_by_max_region_in_mask*)

Creates a video visualizing a given sequence of segmentation mask.

Parameters:

obj_info information on current segmentation

picture_info information on picture size

scale_by_max_region_in_mask switch toggling scaling of gray values: TRUE indicates that the gray values should be scaled w.r.t. to the max region in current mask; FALSE indicates that gray values should be scaled w.r.t. to the max region in whole sequence

Definition at line 4197 of file object_segmentation.c.

References object_info_struct::frame_count, MALLOC, picture_info_struct::picture_size, and object_info_struct::previous_max_number_of_regions.

Referenced by double_labstar_object_segmentation_exit().

Literaturverzeichnis

[1] Mezaris, Vasileos; Kompatsiaris, Ioannis; Strintzis, Michael G.; "Video Object Segmentation Using Bayes-Based Temporal Tracking and Trajectory-Based Region Merging"; IEEE Transactions on Circuits and Systems for Video Technology, Vol. 14, No. 6, June 2004; S. 782-795

[2] Jain, R.; Kasturi, R.; Machine Vision. New York: McGraw-Hill, 1995

[3] Lindbloom, Bruce; "Useful Color Equations"; http://www.brucelindbloom.com/; 13.10.2004

[4] Yu, Tianli; Zhang, Yujin; "Retrieval of video clips using global motion information"; Electronic Letters, Vol. 37, No. 14, 5. Juli 2001; S. 893-895

[5] Philpot, Wiliam; "Digital Image Processing"; http://ceeserver.cee.cornell.edu/wdp2/cee615/Monograph/615_08_UnSuper_Class_rev01.pdf; 18.10.2004; S. 8-4f

[6] VQEG-Sequenz: VQEG_canue; Canoeing on a wild river

[7] Sequenz: Table-Tennis; Auf CIF-Format mittels Spline-Scale umgerechnet

[8] Burden, Richard L.; Faires, Douglas J.; "Numerical Analysis"; Brooks/Cole; Pacific Grove; 2001; 7th Edition; S. 352