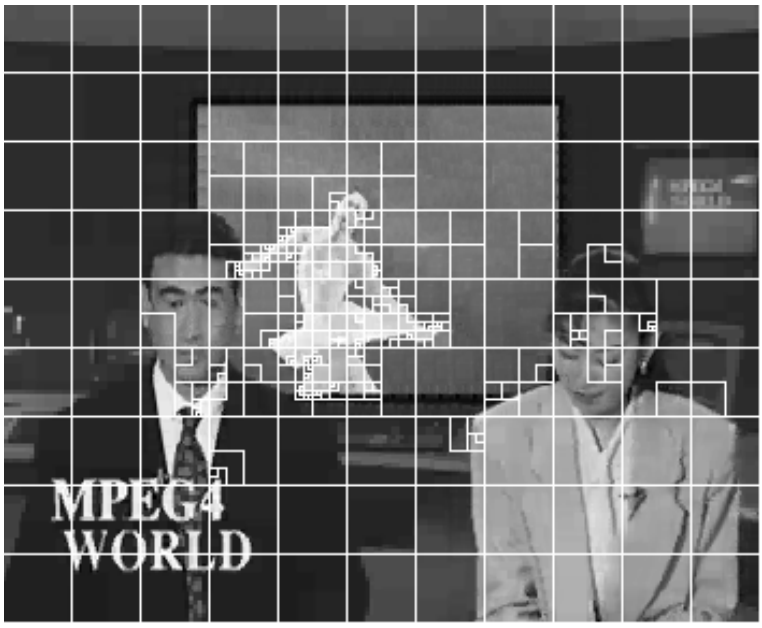
Lehrstuhl für Nachrichtentechnik
Universität Erlangen-Nürnberg
Prof. Dr.-Ing. B. Girod

Studienarbeit



Untersuchung von Algorithmen zur Bestimmung von Quadtree-Strukturen bei der Videocodierung mit Blöcken variabler Größe

Betreuung: Dipl.-Ing. Thomas Wiegand
Bearbeitung: Markus Helmut Flierl

03. März 1997

Studienarbeit

für

Markus Flierl

Untersuchung von Algorithmen zur Bestimmung von Quadtree-Strukturen bei der Videocodierung mit Blöcken variabler Größe

Blockbasierte Videocoder mit bewegungskompensierter Prädiktion stellen derzeit die erfolgreichsten Algorithmen zur Quellencodierung bewegter Bilder dar. Daher finden sich diese Algorithmen in Videocodierstandards wie MPEG, H.261 und H.263 wieder. Bei der Entwicklung des neuen Standards MPEG-4 und der Weiterentwicklung von H.263 wird eine Erweiterung der blockbasierten Bewegungskompensation auf Blöcke variabler Größe in Betracht gezogen, welche sich durch Quadtree-Strukturen effizient implementieren läßt. Um eine vollständige Suche zur Bestimmung der optimalen Quadtree-Struktur aus Rechenzeitgründen zu umgehen, werden verschiedene Algorithmen vorgeschlagen.

Herr Flierl erhält die Aufgabe, unterschiedliche Algorithmen zur Bestimmung von Quadtree-Strukturen bei der Videocodierung mit Blöcken variabler Größe zu untersuchen und zu vergleichen. Dabei sollen „bottom-up“- und „top-down“-Strategien zur Bestimmung der Quadtree-Strukturen betrachtet werden. Der Entwurf der Quadtree-Coder soll in einen aus der Vektor-Quantisierung mit Entropie-Nebenbedingung bekannten iterativen Algorithmus eingebettet werden. Die untersuchten Verfahren sollen anhand von PSNR-Messungen und visueller Bildqualität der decodiertern Bilder und der Recheneffizienz des Coders evaluiert und miteinander verglichen werden.

Die Implementierung der Algorithmen erfolgt in der Programmiersprache C. Besonderer Wert wird auf strukturierte Programmierung und ausführliche Programmdokumentation gelegt.

Erklärung

„Ich versichere, daß ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und daß die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.“

Erlangen, 03. März 1997

An dieser Stelle möchte ich all denen danken, die mich bei der Anfertigung dieser Studienarbeit unterstützt haben. Vor allem möchte ich mich bei meinen Eltern bedanken, die mir ihre Unterstützung jederzeit zugesichert haben. Meinem Studienkollegen Mostafa Elsayed danke ich vor allem für manche persönliche Ratschläge. Für die Durchführung der umfangreichen Simulationen wurde ich von U. Horn und B. Westrich unterstützt. Für ein tieferes Verständnis der fraktalen Codierung bin ich Prof. Cofer dankbar. Im besonderen bin ich Thomas Wiegand und Klaus Stuhlmüller für ihre fachliche Betreuung und stete Diskussionsbereitschaft zu Dank verpflichtet.

Stelle dir also mit Stärke vor, was ein Sterblicher wäre, rein genug, verständig genug, fein und zähe genug, überdies mächtig ausgerüstet von Minerva, um bis an den Rand seines Wesens durchzudenken, um bis an den Rand der Wirklichkeit, diese seltsame Annäherung der sichtbaren Formen mit der hinschwindenden Ansammlung sich vollendender Töne; bedenke, zu welchem heimlichen und allgemeinen Urgrund er vordringen würde, an welchem köstlichen Punkt er anlangen müßte; ...

— *Paul Valéry, Eupalinos*

Inhaltsverzeichnis

Zusammenfassung

In dieser Arbeit wird ein Bewegungs- und Intensitätsmodell für Blöcke variabler Größe untersucht. Dieses Modell dient zur Codierung von Videosequenzen und erlaubt die Reduktion der zeitlichen Korrelation aufeinanderfolgender Bilder. Es werden Algorithmen zur Optimierung von Baumstrukturen analytisch untersucht und Simulationsergebnisse für Pruning- und Growing-Algorithmus anhand von Testsequenzen gezeigt. Zur Optimierung des Coders/Decoders nach dem Rate-Distortion-Kriterium erläutern wir einen iterativen Algorithmus. Wir analysieren den Codec-Entwurf in Abhängigkeit von der Bitverteilungsstrategie (Pruning- oder Growing-Algorithmus) und variieren das Bewegungs- und Intensitätsmodell um Aufschluß über dessen Eigenschaften zu erhalten.

Liste der verwendeten Formelzeichen und Abkürzungen

Signaldarstellungen

$\mathbf{X}$	Videoquelle
$\mathbf{x}$	Videosequenz im Original
$\mathbf{Y}$	Videosenke
$\mathbf{y}$	Rekonstruierte Videosequenz
$\mathbf{C}$	Code
$\mathbf{c}$	Codesequenz
$\mathbf{U}$	Modell
$\mathbf{u}$	Parametersequenz
$\mathbf{V}$	Quantisiertes Modell
$\mathbf{v}$	Quantisierte Parametersequenz
$\mathbf{I}$	Index
$\mathbf{i}$	Indexsequenz

Funktionen

$F_{\mathbf{X}}(\mathbf{x})$	Verteilungsfunktion der Zufallsvariablen $\mathbf{X}$
$f_{\mathbf{X}}(\mathbf{x})$	Wahrscheinlichkeitsfunktion der Zufallsvariablen $\mathbf{X}$
C	Komplexität
$Q(\mathbf{X})$	Quantisierung der Zufallsvariablen $\mathbf{X}$
$\rho(\mathbf{U}, \mathbf{V})$	Fehlermaß zwischen Zufallsvariable $\mathbf{U}$ und $\mathbf{V}$
ϕ	Coder
ψ	Decoder
α	Abbildung des Modells auf den Index
β	Abbildung des Index auf das quantisierte Modell
γ	Abbildung des Index auf den Code

Funktionale

D	Distortion-Funktional
H	Entropie-Funktional

I	Mittlere Information
J	Lagrange-Funktional
R	Raten-Funktional

Operatoren

$E\{\cdot\}$	Erwartungswert-Operator

Mengen

T	Baum, Menge von Knoten
$\tilde{T}$	Endknoten des Baumes T
T^p	Verjüngter Teilbaum von T
T_t	Zweig von T am Knoten t

Symbole

λ	Lagrange-Parameter

Abkürzungen

CIF	Common Intermediate Format
CLGA	Chou Lookabaugh Gray Algorithm
ECVQ	Entropy Constrained Vector Quantization
FLC	Fix Length Code
GLA	Generalized Lloyd Algorithm
MC	Motion Compensation
ME	Motion Estimation
MIC	Motion and Intensity Compensation
MIE	Motion and Intensity Estimation
MIM	Motion and Intensity Model
PSNR	Peak Signal to Noise Ratio
QCIF	Quarter Common Intermediate Format
RGA	Riskin Gray Algorithm
VBS	Variable Block Size
VLC	Variable Length Code
VQ	Vector Quantization
WUVQ	Weighted Universal Vector Quantization

Abbildungsverzeichnis

Kapitel 1

Einleitung

Derzeit verbreitete Algorithmen zur Codierung von Videosequenzen weisen die grundlegende Struktur eines hybriden Coders mit bewegungskompensierender Prädiktion auf. Ein hybrider Coder verbindet einen DPCM-Algorithmus entlang einer Bewegungstrajektorie des Sequenzinhaltes mit einer Fehlerbildcodierung [1]. Die bewegungskompensierende Prädiktion wird in ihren grundlegenden Eigenschaften in [1],[2],[3] untersucht. Bei der Entwicklung des Standards MPEG-4 wird in [4] eine Erweiterung auf eine Bewegungs- und Intensitätskompensation mit Blöcken variabler Größe vorgeschlagen. Dieser Vorschlag realisiert einen DPCM-Algorithmus entlang einer Bewegungs- und Intensitätstrajektorie des Sequenzinhaltes [5],[6]. Die Fehlerbildcodierung des Hybridcoders entfällt dabei.

Die Bewegungs- und Intensitätskompensation kann auch als ortsvariante Abbildung des vorherigen auf das aktuelle Bild betrachtet werden. Zur örtlichen Strukturierung der Abbildung wird ein Quadtree überlagert, dessen Knoten die Blöcke variabler Größe adressieren. Das Problem der Bewegungs- und Intensitätsschätzung bzw. das der Quadtree-Konstruktion wird als eine Einheit betrachtet. Abgesehen von der zusätlichen gemeinsamen Intensitätsschätzung wird die gemeinsame Optimierung bereits in [7],[8],[9] vorgeschlagen.

Zur Optimierung der Quadtree-Strukturen werden in [10],[7],[8],[11] Pruning- bzw. Growing-Algorithmen vorgeschlagen, die Rate-Distortion-optimal sind. Die Optimierungsalgorithmen sind natürliche Strategien, die die Baumstruktur Schritt für Schritt von den Endknoten bis zur Wurzel bzw. von der Wurzel bis zu den Endknoten bestimmen. Die Optimierung mit Hilfe der Rate-Distortion-Theorie ([12],[13]) ergibt ein Kommunikationssystem, daß bei vorgegebener Datenrate die Qualität der Übertragung maximiert. Die Optimierung der Bitverteilung mit Raten-Nebenbedingung erfolgt mit der Lagrange-Formulierung [14].

Die Optimierung der Bitverteilung erfolgt bei gegebenen Quantisierern mit Codes variabler Länge. Der Entwurf der Quantisierer mit Raten-Nebenbedingung impliziert hingegen die gemeinsame Optimierung der Bitverteilung, der Quantisierer und der Codes variabler Länge. (Quantisierer und Code werden im Codebuch zusammengefaßt.) Zur Lösung dieses nichtlinearen Optimierungsproblems wird ein iterativer Algorithmus verwendet, der für die Vektorquantisierung mit Entropie-Nebenbedingung [15] bekannt ist. Dieser ist eine erweiterte Variante des *Generalized Lloyd Algorithm* [16].

Für die Untersuchung von Algorithmen zur Bestimmung von Quadtree-Strukturen bei der Videocodierung mit Blöcken variabler Größe stellen wir in **Kapitel 2** die Architektur des Videocodecs vor. Im Detail wird das Bewegungs- und Intensitätsmodell erläutert, die Codestruktur veranschaulicht und die örtliche Prädiktion der Bewegungsvektoren erklärt. Die Simulationsergebnisse zeigen das verbesserte Rate-Distortion-Verhalten im Vergleich zu einem hybriden Codec nach dem H.263 Standard [17] bei niedrigen Datenraten.

Kapitel 3 diskutiert die Optimierung der Bitverteilung. Nach kurzer Erläuterung des Rate-Distortion-Problems wird der Pruning- bzw. Growing-Algorithmus zur Optimierung von Baumstrukturen abgeleitet. Die Simulationsergebnisse vermitteln einen Eindruck vom Rate-Distortion-Verhalten des suboptimalen Growing-Algorithmus.

Kapitel 4 behandelt den Entwurf der Quantisierer mit Raten-Nebenbedingung, d.h. den Algorithmus, der die optimalen Quantisierer für gegebene Raten-Nebenbedingung iterativ bestimmt. Wir schätzen ein Codebuch, d.h. entwerfen die Quantisierer, aufgrund einer Trainingsmenge und verifizieren die Konvergenz des Design-Algorithmus.

Kapitel 5 untersucht das hierarchische Bewegungs- und Intensitätsmodell unter Anwendung der zu analysierenden Algorithmen. Wir zeigen Ergebnisse für suboptimale und vereinfachte Baumstrukturen, variieren die maximal zulässige Baumtiefe und die örtliche Auflösung der Trainingssequenzen.

Kapitel 6 diskutiert mögliche Variationen des Bewegungs- und Intensitätsmodells und mögliche Formen der Quantisierung des Modells.

Im **Anhang** wird die implementierte Software erläutert, die Simulationsbedingungen festgehalten und mathematische Modelle zur Referenz angegeben.

Kapitel 2

Architektur des Videocodecs

Für die Untersuchung von Algorithmen zur Bestimmung von Quadtree-Strukturen bei der Videocodierung mit Blöcken variabler Größe wurde ein Videocodec realisiert, der eine Bewegungskompensation von Segmenten variabler Größe ermöglicht. Die grundlegenden Konzepte dieses Codecs wurden aus [4] übernommen. In diesem Kapitel wird das zugrundeliegende Modell der Bewegungs- und Intensitätskompensation mit Segmenten variabler Größe, die Struktur der Codesequenz und die örtliche Prädiktion der Bewegungsvektoren erläutert. Detaillierte Implementierungsaspekte sind im **Anhang B** aufgeführt.

2.1 Bewegungs- und Intensitätsmodell

Derzeitig standardisierte Algorithmen zur Videocodierung bei niedrigen Bitraten, wie z.B. H.263 [17], realisieren hybride Codierungsschemata, die sich aus bewegungskompensierender Prädiktion und Intra-Bild-Codierung des Prädiktionsfehlers zusammensetzen **(Abbildung 2.1)**. Der *Prediction Mode (P-Mode)* des H.263-Standards [17] teilt z.B. das aktuelle Bild einer Videosequenz in Blöcke konstanter Größe ein. Durch ein Schätzverfahren werden diesen Blöcken Bewegungsvektoren zugeordnet, die die relative Verschiebung des Blocks aus dem vorherigen Bild in das aktuelle Bild beschreiben. Das so entstandene geschätzte Bild wird vom Originalbild subtrahiert. Das ebenfalls in Blöcke partitionierte Prädiktionsfehlerbild wird blockweise einer diskreten Cosinus-Transformation unterworfen und die sich daraus ergebenden Koeffizienten quantisiert und codiert.

Im Vergleich zu einem Hybrid-Codec unterscheidet sich die Architektur des realisierten Codecs in einem signifikanten Punkt. Für jeden Block variabler Größe wird neben der Bewegungskompensation eine Intensitätskompensation durchgeführt. Diese Variation hat zur Folge, daß die Energie des Prädiktionsfehlerbildes derart gering ist, daß eine Intra-Bild-Codierung überflüssig wird. Ein zusätzliches Feature des VBS-Codec ist die Codierung eines Segments, das sich aus Blöcken zusammensetzt. Diese Form der Codierung führt zu einem verbesserten Rate-Distortion-Verhalten des Codecs, da dadurch Blöcke gemeinsam codiert werden können. Im folgenden wird nun das Bewegungs- und Intensitätsmodell für Segmente variabler Größe näher erläutert.

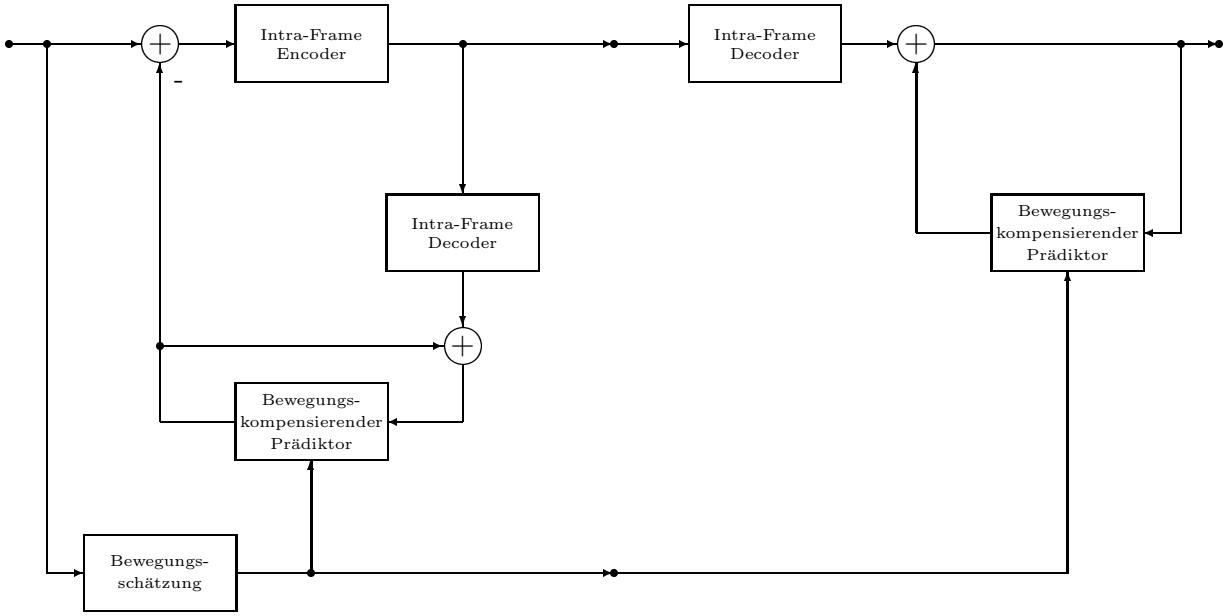


Abbildung 2.1: *Bewegungskompensierender Hybrid-Codec*

2.1.1 Segmentierung

Die Anordnung von Bildpunkten zu Segmenten erlaubt eine gemeinsame Codierung von Blöcken. Zur konkreten Erläuterung beschränken wir uns auf Videosequenzen bestehend aus QCIF-Bildern im 4:2:0-YUV-Format. Die Luminanz-Komponente besteht aus 176×144, die beiden Chrominanz-Komponenten aus 88×72 Pixel, d.h. die U- und V-Komponenten sind im Vergleich zur Y-Komponente jeweils in beiden Dimensionen um den Faktor Zwei unterabgetastet.

Für den VBS-Codec sind Blöcke der Größe $2^\nu \times 2^\nu$ mit $\nu = 1, 2, \ldots$ zulässig. Die Kompensation eines 1×1-Blocks wurde nicht realisiert. Für das QCIF-Bild sind somit Y-Blöcke der Größe 16×16, 8×8, 4×4 und 2×2 und U- bzw. V-Blöcke der Größe 8×8, 4×4 und 2×2 möglich.

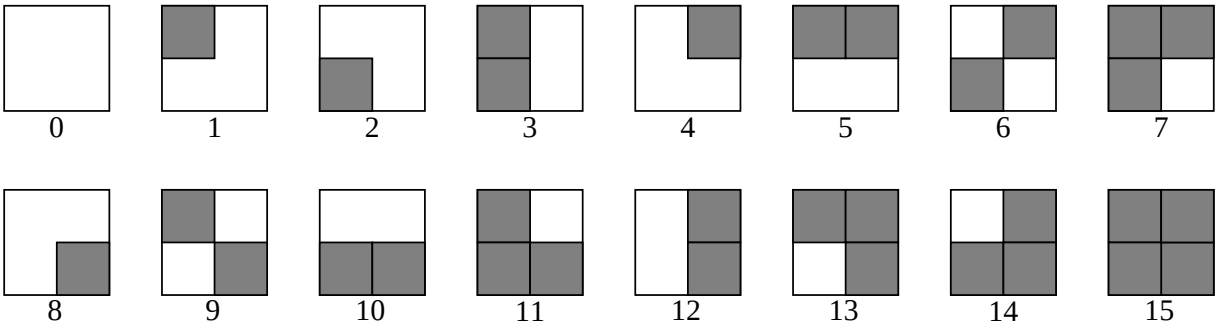


Abbildung 2.2: *Sechzehn verschiedene Möglichkeiten vier Teilblöcke zu kombinieren. Die markierten Teilblöcke gehören nicht zum Segment und können weiter zerlegt werden.*

Die Blöcke sind hierarchisch angeordnet, d.h. ein $2^{\nu+1} \times 2^{\nu+1}$-Block wird in vier $2^\nu \times 2^\nu$ Blöcke zerlegt. Diese vier Teilblöcke können nun beliebig angeordnet werden, d.h. es

können Teilblöcke zu einem Segment zusammengefaßt und gemeinsam codiert werden. **Abbildung 2.2** zeigt die 2^4 Möglichkeiten der Anordnung. Die markierten Teilblöcke gehören nicht zum Segment und werden unabhängig von diesem codiert.

2.1.2 Segmentbasiertes Bewegungs- und Intensitätsmodell

Die *Segmentierung*, d.h. die örtlichen Diskretisierung, impliziert, daß den Bildpunkten eines Segments ein einziges Tripel bestehend aus quantisierter örtlicher Verschiebung und Intensitätsdifferenz zugeordnet werden kann.

Abbildung 2.3 verdeutlicht die Zusammenfassung von Bildpunkten zu einem Segment S. Diesem wird eine gemeinsame örtliche Verschiebung $(\mathbf{\Delta m}, \mathbf{\Delta n})$ und Intensitätsdifferenz $\mathbf{q}$ zugewiesen. Wir kennzeichnen die örtliche Verschiebung und die Intensitätsdifferenz als Funktion der Bildpunktposition (m, n, k).

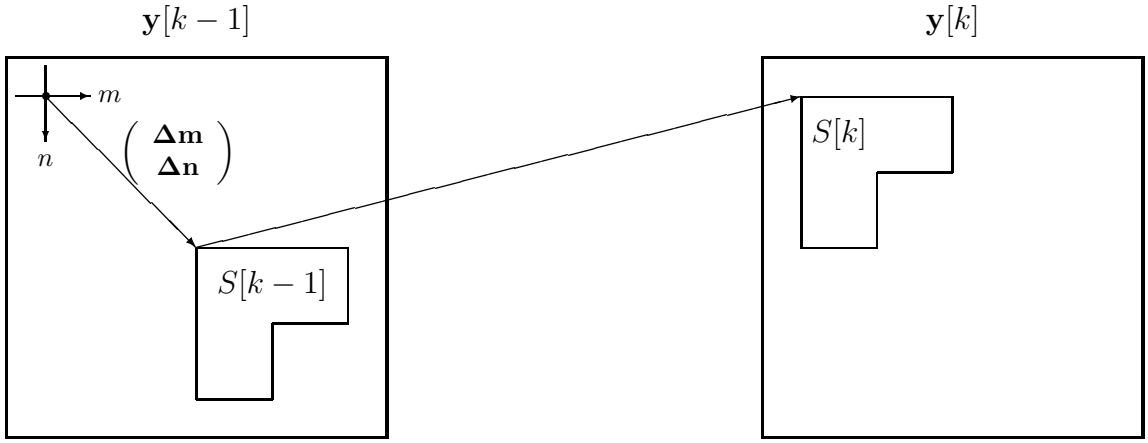


Abbildung 2.3: *Translation eines Segments vom rekonstruierten Bild $k-1$ zum rekonstruierten Bild k mit Intensitätskorrektur*

Das Korrespondenzproblem, d.h. die Zuordnung der Segmente im vorherigen und im aktuellen Bild, lösen wir hier durch *Rückwärtsschätzung* [18]. Dabei definieren wir die Orientierung der Bewegungsvektoren vom Zeitpunkt k zu $k-1$ und das Vorzeichen der Intensitätsdifferenz vom Zeitpunkt $k-1$ zu k. Die Rückwärtsschätzung bei der Codierung wird durch eine *Vorwärtskompensation* bei der Decodierung ergänzt [18] und definiert somit das *Bewegungs- und Intensitätsmodell* wie in **Gleichung 2.1** beschrieben.

$$\mathbf{y}[m,n,k] = \mathbf{y}[m+\mathbf{\Delta m}[m,n,k], n+\mathbf{\Delta n}[m,n,k], k-1]+\mathbf{q}[m,n,k] \qquad (m,n) \in S[k] \quad (2.1)$$

Bei der Vorwärtskompensation betrachten wir das Segment $S[k-1]$ im rekonstruierten Bild. Die Position erhalten wir durch die Verschiebung des Segments $S[k]$ um den örtliche Versatz $(\mathbf{\Delta m}, \mathbf{\Delta n})$. Die Intensitäten der Bildpunkte in Segment $S[k-1]$ werden nun nach Addition der Intensitätsdifferenz den Bildpunkten im Segment $S[k]$ zugeordnet.

Die Rückwärtsschätzung wird mit der Methode des *Segment-Matching* realisiert um die örtliche Verschiebung und Intensitätsdifferenz zu bestimmen [18]. Mit einem Optimalitätskriterium und einer Suchstrategie wird die beste Übereinstimmung eines Segments

im vorherigen rekonstruierten Bild angestrebt. Diese Suche ist für alle Segmente des aktuellen zu rekonstruierenden Bildes durchzuführen.

Damit bei der Vorwärtskompensation jeder Bildpunkt kompensiert werden kann, muß das aktuelle zu rekonstruierende Bild derart in Segmente eingeteilt werden, daß dieses vollständig abgedeckt ist.

2.1.3 Kompensation mit Segmentüberlappung

Die oben beschriebene Methode der nicht-überlappenden Segmentierung führt vor allem bei größeren Segmenten zu Blockeffekten. Bei diesen visuellen Effekten empfindet der Betrachter die sich bewegenden Blockkanten als störend. Um diese Blockeffekte zu vermindern, wird in [4] eine Methode vorgeschlagen, mit der neun gewichtete Bildpunkte anstatt einem kompensiert werden. Die Maske zur Gewichtung eines kompensierten Bildpunktes wird in **Abbildung 2.4** dargestellt und bei der Kompensation mit Segmentüberlappung verwendet.

$$\begin{array}{ccc} \frac{2}{25} & \frac{3}{25} & \frac{2}{25} \\ \frac{3}{25} & \boxed{\frac{5}{25}} & \frac{3}{25} \\ \frac{2}{25} & \frac{3}{25} & \frac{2}{25} \end{array}$$

Abbildung 2.4: *Maske zur Gewichtung eines kompensierten Bildpunktes*

Das Bewegungs- und Intensitätsmodell nach **Gleichung 2.1** wird nun durch die Maske $M[m,n]$ erweitert. Der Bezugspunkt $M[0,0]$ ist in **Abbildung 2.4** markiert. Die Intensität des kompensierten Bildpunktes an der Position (m,n) setzt sich aus neun gewichteten Bildpunktintensitäten im vorherigen Bild zusammen.

$$\mathbf{y}[m,n,k] = \sum_{u,v} M[u,v] \tag{2.2}$$
$$\Big\{\mathbf{y}[m+\mathbf{\Delta m}[m+u,n+v,k], n+\mathbf{\Delta n}[m+u,n+v,k], k-1] + \mathbf{q}[m+u,n+v,k]\Big\}$$

Erhalten die acht Nachbarn des kompensierten Bildpunktes mit der Position (m,n) identische Kompensationswerte (örtliche Verschiebung, Intensitätsdifferenz), so addieren sich die Gewichte $M[u,v]$ zu 1 und das Resultat entspricht einer Kompensation ohne Maske nach **Gleichung 2.1**.

$$\mathbf{\Delta m}[m+u,n+v] = \mathbf{\Delta m}[m,n] \qquad \forall u,v \in \{-1,0,1\} \tag{2.3}$$
$$\mathbf{\Delta n}[m+u,n+v] = \mathbf{\Delta n}[m,n] \qquad \forall u,v \in \{-1,0,1\} \tag{2.4}$$
$$\mathbf{q}[m+u,n+v] = \mathbf{q}[m,n] \qquad \forall u,v \in \{-1,0,1\} \tag{2.5}$$

Die Effekte der gewichteten Kompensation treten deshalb nur an den Blockrändern auf. Zum Beispiel kann ein 4×4-Block mit der Maske in **Abbildung 2.5** gewichtet werden.

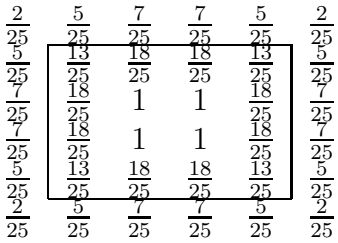


Abbildung 2.5: *Maske zur Gewichtung eines kompensierten* 4×4*-Blocks*

An den Bildrändern existieren nicht alle acht Bildpunktnachbarn. Deshalb sind in den Bildecken und an den Bildkanten modifizierte Randmasken nach **Abbildung 2.6** zu verwenden. Die fehlenden Randmasken sind durch Spiegelung an horizontalen und vertikalen Achsen zu erhalten.

Abbildung 2.6: *Randmasken für Bildecken und Bildkanten.*

Zur optimalen Bewegungs- und Intensitätskompensation mit überlappenden Segmenten sind z.B. Schätzverfahren nach [19] anzuwenden. Für die vorliegenden Untersuchungen wurde die Segmentüberlappung bei der Schätzung nicht berücksichtigt und somit eine suboptimale Kompensation mit überlappenden Segmenten realisiert.

2.2 Die Codestruktur

Die Codesequenz, die der Coder aus einer Videosequenz erzeugt, gliedert sich hierarchisch in Bild-, Makroblock- und Quadtree-Code. Das folgende Kapitel erläutert diese Codestruktur in Verbindung mit dem Codebuch, das die Codesequenz konstituierenden Codeworte enthält.

2.2.1 Der Bildcode

Die in Zeitrichtung diskretisierte Videosequenz wird bildweise codiert. Dazu wird jedes Bild in 99 Makroblöcke eingeteilt. Der Makroblockcode wird zeilenweise mit dem linken oberen Makroblock beginnend und mit dem rechten unteren Makroblock endend in die Codesequenz eingetragen. Da nur dieser Bildcode zugelassen wird, ist kein Bildstrukturcode explizit in die Codesequenz einzufügen.

2.2.2 Der Makroblockcode

Der Makroblockstrukturcode (COD) besteht nur aus einem Bit. Ist dies Null, so enthält die Codesequenz keine weitere Information über den aktuellen Makroblock. Der Decoder interpretiert dies so, als sei die zur Decodierung notwendige Information in der bereits decodierten Teilcodesequenz enthalten.

Ist das COD-Bit gesetzt, so ist neben der Information der bereits decodierten Teilcodesequenz zusätzlich die Information des folgenden Quadtree-Codes zu verwenden, um den aktuellen Makroblock zu decodieren. Dem gesetzten COD-Bit folgt der Quadtree-Code der Y-, U- und V-Komponente in der angegebenen Reihenfolge.

Die Größe des Makroblocks ist von der Auflösung der codierten Videosequenz abhängig. CIF-Bilder bestehen aus 32×32 Y-Makroblöcken und 16×16 U- und V-Makroblöcken. Die Makroblöcke der QCIF-Bilder sind in beiden Dimensionen um den Faktor Zwei kleiner.

2.2.3 Quadtree-Code

Der Y-, U- und V-Makroblock wird jeweils durch die Codierung in eine allgemeine Quadtree-Struktur mit bis zu vier Zweigen pro Knoten zerlegt **(Abbildung 2.7)**. Jeder Knoten dieses Baumes enthält die Information über Segmentform, örtliche Verschiebung und Intensitätsdifferenz. Der Quadtree-Code ist nun eine Beschreibung des Quadtrees, d.h. der Quadtree-Code entsteht durch eine Anordnungsregel, nach der die Knoteninformationen des Baumes sequentiell angeordnet werden. Die verwendete Anordnungsregel zur Erzeugung des Quadtree-Codes lautet wie folgt: „Jeder Zweig (Teilblock) des aktuellen Knotens (Blocks) wird vor dem nächsten Zweig des aktuellen Knotens beschrieben (*Depth-First*).“

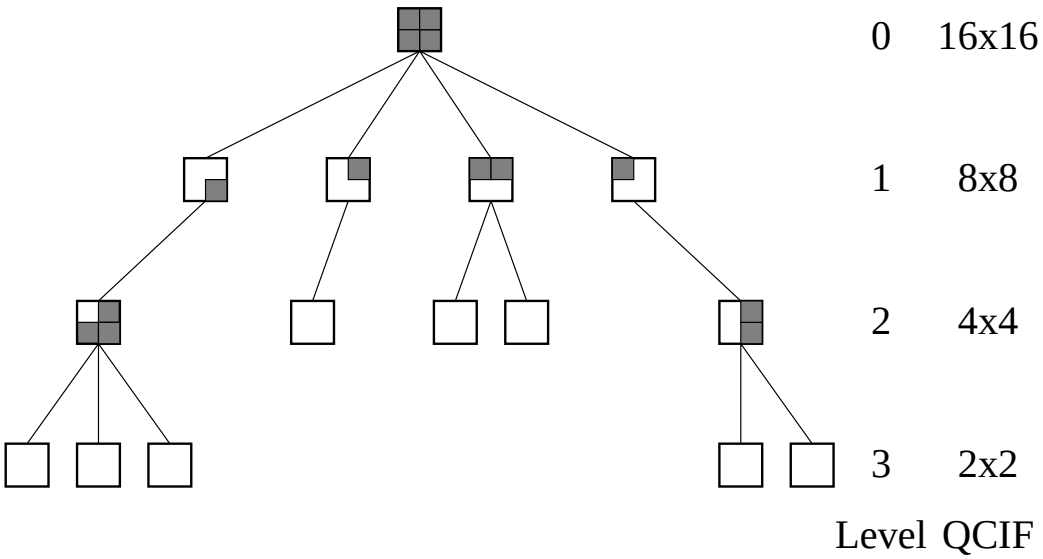


Abbildung 2.7: *Allgemeine Quadtree-Struktur mit bis zu vier Zweigen pro Knoten*

Abbildung 2.8 verdeutlicht die Anordnungsregel zur Bestimmung des Quadtree-Codes an einem Beispiel. Der Level-0-Block wird mit Segmentform 5 und Quantisierer Q(0) (örtliche Verschiebung und Intensitätsdifferenz) codiert. Anschließend wird mit dem Code der Teilblöcke in der Reihenfolge 0-1-2-3 unter Berücksichtigung der Regel „Depth-First“ fortgefahren. Der Teilblock 0 auf dem Level 1 wird mit Segmentform 9 und Quantisierer Q(1,0) codiert. Bevor nun mit dem Teilblock 2 auf dem Level 1 fortgefahren wird, werden die Teilblöcke 0 und 3 auf dem Level 2 mit jeweils der Segmentform 0 und Quantisierer Q(2,0) und Q(2,3) codiert. Schließlich folgt nun nach der Regel „Depth-First“ Teilblock 2 auf Level 1 mit Segmentform 4 und Q(1,2) und der Teilblock 2 auf Level 2 mit Segmentform 0 und Q(2,2).

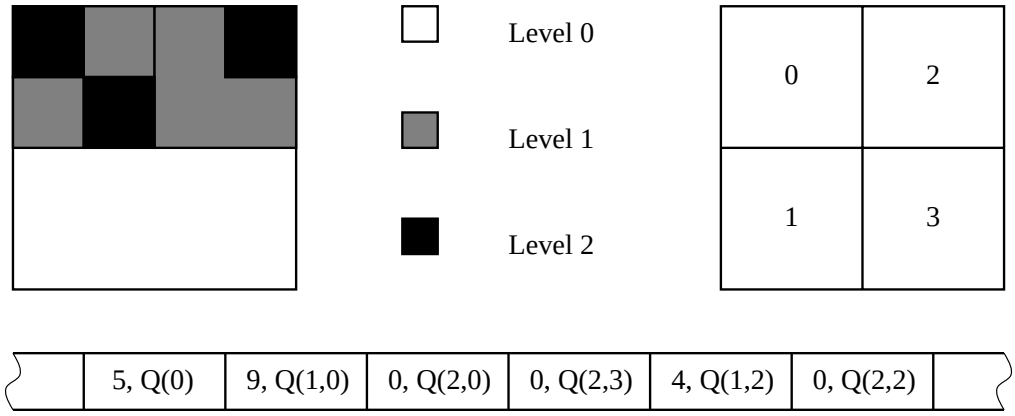


Abbildung 2.8: *Beispiel zur Anordnungsregel zur Bestimmung des Quadtree-Code*

Nach der Anordnungsregel ist für jede Farbkomponente auf jeder Hierarchiestufe ein Codewort für die Segmentform, örtliche Verschiebung und Intensitätsdifferenz vorgesehen. Es gibt jedoch zwei Ausnahmen. Auf der letzten Hierarchiestufe ist kein Codewort für die Segmentform im Quadtree-Code einzufügen, da diese Knoten immer Endknoten sind und somit die Segmentform Null besitzen. Weiterhin möchte man den Bewegungsvektor des Y-Makroblocks den U- und V-Makroblöcken vererben. Die Bewegungsvektoren entsprechen den örtlichen Verschiebungen nur dann, wenn bei der Schätzung nur eine Intensitätsdifferenz von Null zugelassen wird. Für diesen Fall muß somit die Intensitätsdifferenz der Makroblöcke für alle Farbkomponenten nicht codiert werden. Die Bewegungsvektoren der U- und V-Makroblöcke ergeben sich aus dem Bewegungsvektor des Y-Makroblocks durch eine Integer-Division mit Zwei und werden somit nicht durch ein Codewort im Quadtree-Code repräsentiert.

Der Quadtree-Code ist somit eine Sequenz aus Codeworten für die Segmentform, örtliche Verschiebung und Intensitätsdifferenz mit der Syntax nach **Abbildung 2.9**. Diese berücksichtigt die Endknoten- und Vererbungsregel.

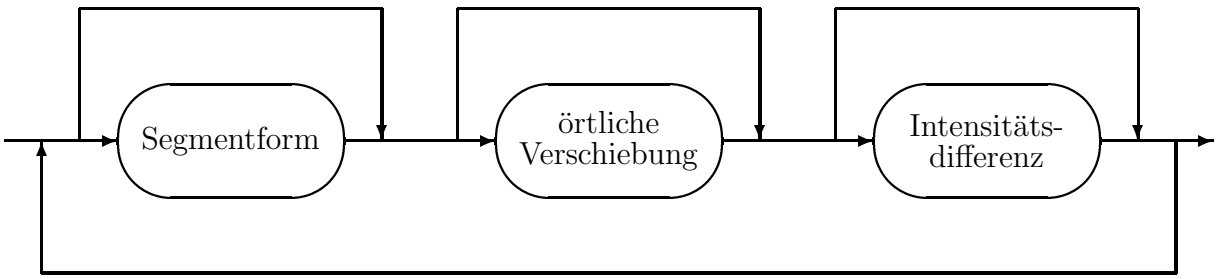


Abbildung 2.9: *Syntax-Diagramm des Quadtree-Codes*

2.2.4 Das Codebuch

Das Codebuch besteht aus individuellen Codebüchern für jede Farbkomponente und Hierarchiestufe. Ein partielles Codebuch enthält je einen Huffman-Code für die Tabelle der möglichen Segmentformen, der codierten örtlichen Verschiebungen und der zugelassenen Intensitätsdifferenzen.

2.3 Örtliche Prädiktion der Bewegungsvektoren

Neben der temporalen Prädiktion im **Abschnitt 2.1** ermöglicht auch die örtliche Prädiktion die Reduktion der Korrelation benachbarter Abtastwerte. Bei der örtlichen Prädiktion der Bewegungsvektoren eines Makroblocks versucht man einen Schätzwert des Bewegungsvektors des aktuellen Makroblocks aus den tatsächlichen Bewegungsvektoren der örtlichen Nachbarn zu bestimmen. Somit muß nur noch die Differenz zwischen geschätztem und tatsächlichem Bewegungsvektor codiert werden. Die tatsächliche örtliche Verschiebung $(\mathbf{\Delta m}, \mathbf{\Delta n})$ setzt sich aus der geschätzten örtlichen Verschiebung $(\mathbf{p}_m, \mathbf{p}_n)$ und der Differenz $(\mathbf{d}_m, \mathbf{d}_n)$ zusammen.

$$(\mathbf{\Delta m}, \mathbf{\Delta n}) = (\mathbf{p}_m, \mathbf{p}_n) + (\mathbf{d}_m, \mathbf{d}_n) \tag{2.6}$$

Abbildung 2.10 veranschaulicht die örtlichen Prädiktion aus den Top-Level Bewegungsvektoren der Nachbarmakroblöcke. Für den aktuellen Makroblock wird jeweils ein Schätzwert $(\mathbf{p}_m, \mathbf{p}_n)$ bestimmt.

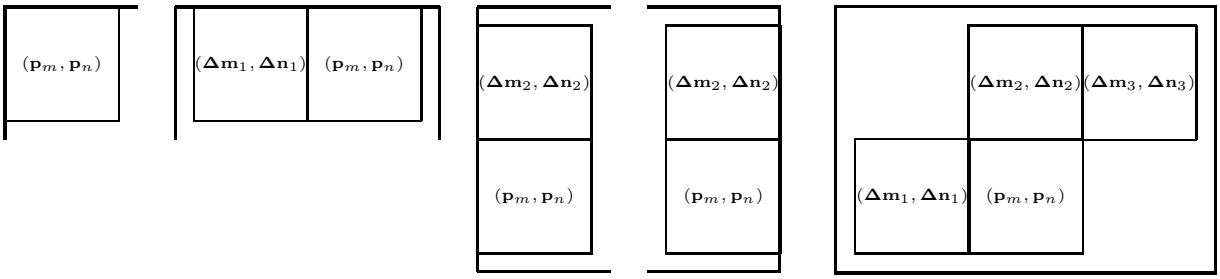


Abbildung 2.10: *Örtliche Prädiktion aus den Top-Level Bewegungsvektoren*

Dabei werden vier verschiedene Fälle unterschieden:

1. Der aktuelle Makroblock ist in der linken oberen Ecke des Bildes. Für diesen Fall wird der Schätzwert auf Null gesetzt:
$$(\mathbf{p}_m, \mathbf{p}_n) = (0, 0).$$
2. Der aktuelle Makroblock befindet sich in der obersten Zeile des Bildes. Der Schätzwert entspricht hier dem tatsächlichen Bewegungsvektor des linken Nachbarn:
$$(\mathbf{p}_m, \mathbf{p}_n) = (\mathbf{\Delta m}_1, \mathbf{\Delta n}_1).$$
3. Der aktuelle Makroblock befindet sich am linken oder rechten Rand des Bildes. Der Schätzwert nimmt hier den tatsächlichen Bewegungsvektor des oberen Nachbarn an:
$$(\mathbf{p}_m, \mathbf{p}_n) = (\mathbf{\Delta m}_2, \mathbf{\Delta n}_2).$$
4. Der aktuelle Makroblock befindet sich im Inneren des Bildes. Der Schätzwert berechnet sich komponentenweise aus dem *Median* [18] der tatsächlichen Bewegungsvektoren der linken und oberen Nachbarn:
$$(\mathbf{p}_m, \mathbf{p}_n) = (Med(\mathbf{\Delta m}_1, \mathbf{\Delta m}_2, \mathbf{\Delta m}_3), Med(\mathbf{\Delta n}_1, \mathbf{\Delta n}_2, \mathbf{\Delta n}_3)).$$

Für den Fall, daß der Makroblock (Top-Level-Block) die Segmentform 15 aufweist und somit kein Bewegungsvektor bestimmt wird, ist dieser auf Null zu setzen.

2.4 Simulationsergebnisse

Es wurden Simulationen für den Vergleich zwischen einem Modell mit Bewegungs- und Intensitätskompensation mit Segmenten variabler Größe (VBS-Codec) und einem Modell mit Bewegungskompensation und Fehlerbildcodierung (TMN 1.6) erstellt. Die Simulationsbedingungen für die folgenden Ergebnisse sind in **Anhang C** erläutert.

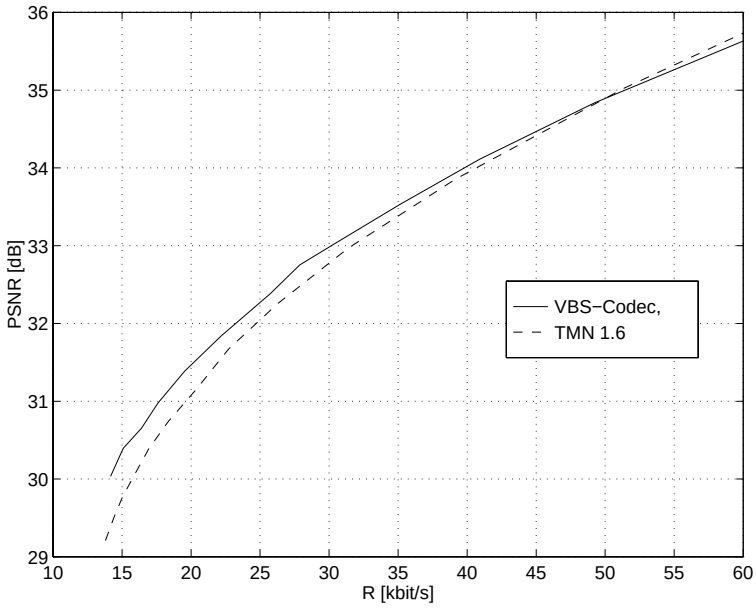


Abbildung 2.11: *Vergleich zwischen VBS-Codec mit Pruning-Algorithmus und TMN 1.6 anhand der Testsequenz „Car Phone“ (QCIF, 7.5 fps, 10 s).*

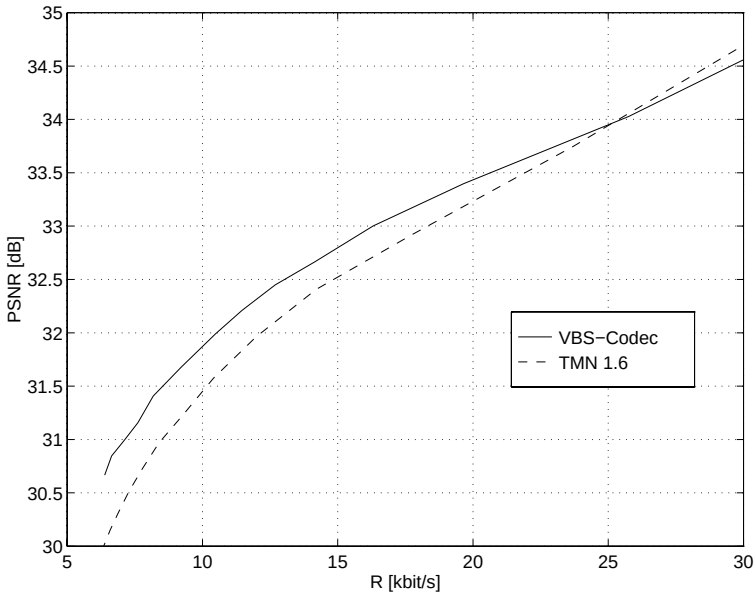


Abbildung 2.12: *Vergleich zwischen VBS-Codec mit Pruning-Algorithmus und TMN 1.6 anhand der Testsequenz „Salesman“ (QCIF, 7.5 fps, 10 s).*

Abbildungen 2.11 und 2.12 zeigen das Rate-Distortion-Verhalten des VBS-Codec bzw. des TMN 1.6 für die Testsequenzen in QCIF-Auflösung. Dabei wurden die Testsequenzen für verschiedene λ-Werte bzw. Quantisierungsparameter codiert. Für niedrige Bitraten erzielt das Bewegungs- und Intensitätsmodell Gewinne von ca. 0.75 dB. Die Rate-Distortion-Kurven der beiden Verfahren schneiden sich, da der VBS-Codec keinen Intra-Mode zur Verfügung stellt und das erste Bild der codierten Sequenzen für alle Raten identisch ist. Das erste Bild der Sequenz „Car Phone“ erzielt einen $PSNR$-Wert von 35.0 dB, das der

Sequenz „Salesman“ einen $PSNR$-Wert von 33.5 dB. Der VBS-Codec ist nicht in der Lage, den „ruhenden“ Hintergrund ebenso effektiv wie der TMN 1.6 im *Intra-Mode* zu codieren.

Für die Codierung der QCIF-Sequenzen wurde der Pruning-Algorithmus zur Optimierung der Bitverteilung verwendet. Die Bewegungs- und Intensitätskompensation wurde für die Y-Farbkomponente mit Blöcken der Größe 16×16, 8×8, 4×4 und 2×2 bzw. für die U- und V-Farbkomponenten mit Blöcken der Größe 8×8, 4×4 und 2×2 realisiert.

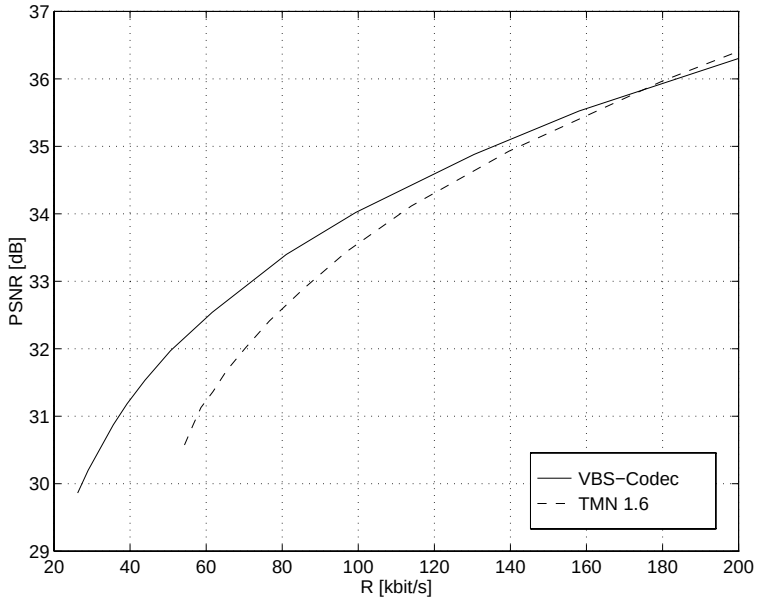


Abbildung 2.13: *Vergleich zwischen VBS-Codec mit Pruning-Algorithmus und TMN 1.6 anhand der Testsequenz „Car Phone“ (CIF, 7.5 fps, 10 s).*

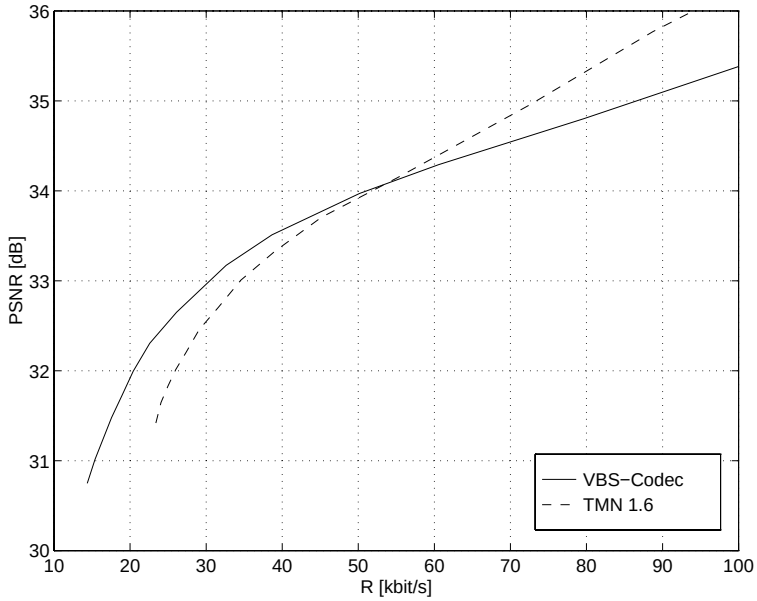


Abbildung 2.14: *Vergleich zwischen VBS-Codec mit Pruning-Algorithmus und TMN 1.6 anhand der Testsequenz „Salesman“ (CIF, 7.5 fps, 10 s).*

Abbildungen 2.13 und 2.14 zeigen das Rate-Distortion-Verhalten des VBS-Codec bzw. des TMN 1.6 für die Testsequenzen in CIF-Auflösung. Dabei wurden die Testsequenzen für verschiedene λ-Werte bzw. Quantisierungsparameter codiert. Die Aussagen für die QCIF-Testsequenzen bezüglich des Intra-Modes treffen auch für die CIF-Testsequenzen zu. Die $PSNR$-Werte der ersten Bilder der CIF-Sequenzen liegen für „Car Phone“ bei

35.9 dB und für „Salesman“ bei 34.7 dB. Für niedrige Bitraten sind bei den untersuchten Testsequenzen Gewinne von 1 - 1.5 dB festzustellen. Bei dieser Auflösung demonstriert das hierarchische Bewegungs- und Intensitätsmodell deutlich seine Überlegenheit.

Der Pruning-Algorithmus optimiert die Bitverteilung auch für die CIF-Sequenzen. Für die Bewegungs- und Intensitätskompensation wurden jedoch für die Y-Farbkomponente Blöcke der Größe 32×32, 16×16, 8×8, 4×4 und 2×2 bzw. für die U- und V-Farbkomponenten Blöcke der Größe 16×16, 8×8, 4×4 und 2×2 zur Codierung herangezogen.

Kapitel 3

Optimierung der Bitverteilung

Das Ziel der angestrebten Quellencodierung von Videosequenzen ist die Elimination von redundanten und irrelevanten Daten bei gegebenem Optimalitätskriterium. Die Reduktion der redundanten Daten erfolgt durch Extraktion der Wiederholungen von Quelleninhalten. Der wiederholungsfreie Inhalt ist unter gegebenen Umständen dem Beobachter zum Teil irrelevant, wird also z.B. vom Beobachter nicht in der Detailfülle benötigt, und ist somit durch ein vom Beobachter zu definierendes Maß zu bestimmen. Im folgenden versuchen wir nun das Problem der Quellencodierung aus dem Blickwinkel der Informationstheorie, d.h. mit Hilfe der Rate-Distortion-Theorie, zu behandeln.

3.1 Das Rate-Distortion-Problem

Für die Untersuchungen verwenden wir ein Modell des Kommunikationssystems nach **Abbildung 3.1**. Die Videosequenz $\mathbf{x}$ wird im Coder auf die Codesequenz $\mathbf{c}$ abgebildet. Für das Modell des Kommunikationssystems nehmen wir einen störungsfreien Übertragungskanal an. Somit kann der Decoder eindeutig aus der Codesequenz $\mathbf{c}$ die rekonstruierte Videosequenz $\mathbf{y}$ erzeugen.

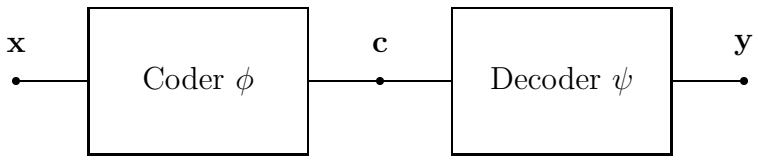


Abbildung 3.1: *Modell des Kommunikationssystems*

Die diskrete Quelle $\mathbf{X}$, die die Videosequenz $\mathbf{x}$ sendet, beschreiben wir durch die Wahrscheinlichkeitsfunktion $f_{\mathbf{X}}(\mathbf{x})$. Dessen Werte entsprechen den Wahrscheinlichkeiten, mit denen die Quelle $\mathbf{X}$ die einzelnen Videosequenzen $\mathbf{x}$ sendet [13]. Analoges gilt für den Code $\mathbf{C}$ und die Senke $\mathbf{Y}$.

$$f_{\mathbf{X}}(\mathbf{x}) = P(\mathbf{X} = \mathbf{x}) \tag{3.1}$$

Die mittlere wechselseitige Information oder kurz die wechselseitige Information, daß die Quelle $\mathbf{X}$ die Videosequenz $\mathbf{x}$ sendet und die Senke $\mathbf{Y}$ die Videosequenz $\mathbf{y}$ empfängt, ist

als der Erwartungswert des Logarithmus des Verhältnisses der Verbundwahrscheinlichkeitsfunktion zu dem Produkt der Wahrscheinlichkeitsfunktionen der Quelle $\mathbf{X}$ und der Senke $\mathbf{Y}$ definiert [13]. Wird der Logarithmus zur Basis Zwei verwendet, so besitzt die Information die Einheit „bit“.

$$I(\mathbf{X},\mathbf{Y}) = E\left\{ld\frac{f_{\mathbf{XY}}}{f_{\mathbf{X}}f_{\mathbf{Y}}}\right\} \tag{3.2}$$

Um die Qualität der Rekonstruktion zu bestimmen, wird eine Vorschrift benötigt, die jeder möglichen Approximation ein Maß zuweist. Wir bezeichnen dieses Maß als *Distortion* [12]. Die Distortion ist ein nicht negativer Wert und für die Untersuchungen im Rahmen dieser Arbeit verwenden wir den Erwartungswert des quadratischen Fehlers.

$$D(\mathbf{X},\mathbf{Y}) = E\left\{\|\mathbf{X}-\mathbf{Y}\|_2^2\right\} \tag{3.3}$$

Für die optimale Quellencodierung ist nach der Rate-Distortion-Theorie [12] bei gegebener Wahrscheinlichkeitsfunktion der Quelle $f_{\mathbf{X}}(\mathbf{x})$ für jede vorgegebene Rate R die Distortion $D(R)$ über die bedingte Wahrscheinlichkeitsfunktion der Senke $f_{\mathbf{Y}|\mathbf{X}}$ zu minimieren.

$$D(R) = \inf_{f_{\mathbf{Y}|\mathbf{X}}}\left\{D(\mathbf{X},\mathbf{Y})\middle| I(\mathbf{X},\mathbf{Y}) \leq R\right\} \tag{3.4}$$

Für den Fall, daß Quelle und Senke nur endlich viele diskrete Videosequenzen enthalten, löst der *Blahut-Algorithmus* das konvexe Optimierungsproblem [15]. Dieser Algorithmus basiert auf der Minimierung des Lagrange-Funktionals, einer Linearkombination aus Distortion und wechselseitiger Information. Der Lagrange-Parameter λ ist dabei eine nicht negative reelle Zahl.

$$J(f_{\mathbf{Y}|\mathbf{X}},\lambda) = D(\mathbf{X},\mathbf{Y}) + \lambda I(\mathbf{X},\mathbf{Y}) \tag{3.5}$$

Das ursprüngliche Optimierungsproblem mit Nebenbedingung (**Gleichung 3.4**) wird durch die Lagrange-Formulierung mit dem Lagrange-Parameter λ somit zu einem Optimierungsproblem ohne Nebenbedingung (**Gleichung** 3.5).

Um technische Realisierungen zu modellieren, beschreiben wir den Coder ϕ als eine Abbildung der Videoquelle $\mathbf{X}$ auf den Code $\mathbf{C}$, den Decoder ψ als eine Abbildung des Codes auf die Videosenke $\mathbf{Y}$. Die Länge der Codesequenz $\mathbf{c}$ ist nun ein Maß für die Information, die über den Kanal übertragen wird. Die mittlere Anzahl der Kanalsymbole in „bit“ einer Codesequenz $\mathbf{c}$ gibt Auskunft über die mittlere Information, die wir auch als mittlere Datenrate oder kurz Rate bezeichnen.

$$R(\mathbf{C}) = E\left\{|\mathbf{C}|\right\} \tag{3.6}$$

Mit Hilfe dieses Modells erhalten wir nun die realisierbare Rate-Distortion-Funktion $\tilde{D}(R)$. Diese bestimmen wir bei gegebener Wahrscheinlichkeitsfunktion der Quelle $f_{\mathbf{X}}(\mathbf{x})$ und in Abhängigkeit der Zielrate R, indem wir die Distortion über alle Coder ϕ und Decoder ψ minimieren.

$$\tilde{D}(R) = \inf_{\phi,\psi}\left\{D(\mathbf{X},\psi\circ\phi(\mathbf{X}))\middle| R(\phi(\mathbf{X})) \leq R\right\} \tag{3.7}$$

Die konvexe Rate-Distortion-Funktion $D(R)$ ist dabei die untere Schranke der nicht zunehmenden und nicht konvexen realisierbaren Rate-Distortion-Funktion $\tilde{D}(R)$ [15]. Die Lagrange-Formulierung des Problems führt nun auf das realisierbare Lagrange-Funktional.

$$\tilde{J}(\phi,\psi,\mathbf{X},\lambda) = D(\mathbf{X},\psi\circ\phi(\mathbf{X})) + \lambda R(\phi(\mathbf{X})) \tag{3.8}$$

Die Minimierung des realisierbaren Lagrange-Funktionals für variierende Lagrange-Parameter ergibt die konvexe Hülle der realisierbaren Rate-Distortion-Funktion [14]; diese stellt die untere Schranke der realisierbaren Rate-Distortion-Funktion dar [10]. **Abbildung 3.2** veranschaulicht diesen Zusammenhang. Die Minimierung des realisierbaren Lagrange-Funktionals liefert nur die gekennzeichneten Punkte auf der konvexen Hülle. Alle so gefundenen Punkte sind somit auch Lösungen des Optimierungsproblems mit Nebenbedingung nach **Gleichung 3.7**.

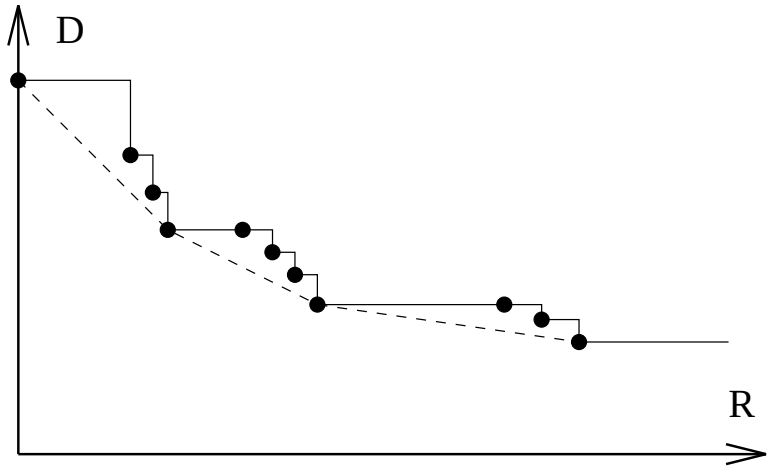


Abbildung 3.2: *Eine typische realisierbare Rate-Distortion-Funktion mit den gekennzeichneten (R, D)-Paaren für eine diskrete Verteilung. Ein Teil der konvexen Hülle ist gestrichelt eingezeichnet.*

Der Lagrange-Parameter λ kann als Steigung der konvexen Hülle interpretiert werden. Dieser Zusammenhang ergibt sich, indem man das totale Differential der Lagrange-Funktion auf Null setzt.

$$dJ = \frac{\partial J}{\partial D}dD + \frac{\partial J}{\partial R}dR = dD + \lambda dR = 0 \quad \Longrightarrow \quad \lambda = -\frac{dD}{dR} \tag{3.9}$$

Wird $\lambda \gg 1$ gewählt, so wird der Ratenterm der Lagrange-Funktionals stärker als die Distortion gewichtet. In **Abbildung 3.2** befinden sich solche Punkte auf der linken Seite der Rate-Distortion-Funktion. Für $0 < \lambda \ll 1$ wird der Ratenterm schwächer als die Distortion gewichtet. Die korrespondierenden Punkte liegen auf der rechten Seite der Rate-Distortion-Funktion.

3.2 Optimierung der Bitverteilung für das MIM

Für die Codierung der Videosequenz verwenden wir das Bewegungs- und Intensitätsmodell (**M**otion and **I**ntensity **M**odel) für Segmente variabler Größe. Für jedes Segment bestimmen wir gemeinsam die optimale Quantisierung Q und Segmentform S durch Minimierung der Lagrange-Kosten. Die Quantisierung $Q = (\mathbf{\Delta m}, \mathbf{\Delta n}, \mathbf{q})$ beschreibt die Wertdiskretisierung der örtlichen Verschiebungen $\mathbf{\Delta m}$ und $\mathbf{\Delta n}$ bzw. der Intensitätsdifferenz $\mathbf{q}$.

$$\hat{J} = \min_{S,Q} J(S, Q) \tag{3.10}$$

Die Lagrange-Kosten summieren sich aus Distortion und den mit λ gewichteten Ratentermen. Die Distortion berechnet sich aus dem quadratischen Fehler der Differenz zwischen der tatsächlichen Intensität $\mathbf{x}[m,n,k]$ zur diskreten Zeit k und der geschätzten verschobenen Intensität $\mathbf{y}[m+\mathbf{\Delta m}, n+\mathbf{\Delta n}, k-1]$ zur diskreten Zeit $k-1$ und der Intensitätsdifferenz. Die Rate addiert sich aus den Längen der Codeworte für die Segmentform, örtliche Verschiebung und Intensitätsdifferenz.

$$J(S,Q) = \sum_{(m,n)\in S} \Big|\mathbf{x}[m,n,k] - \mathbf{y}[m+\mathbf{\Delta m}, n+\mathbf{\Delta n}, k-1] - \mathbf{q}\Big|^2 + \lambda\Big\{\mathbf{l}(S) + \mathbf{l}(\mathbf{\Delta m}, \mathbf{\Delta n}) + \mathbf{l}(\mathbf{q})\Big\} \tag{3.11}$$

Die Segmente variabler Größe sind in einer hierarchischen Struktur angeordnet. Jedes Segment setzt sich aus Segmenten kleinerer Größe zusammen und wird für alle Hierarchiestufen nur aus einem Segmentformenvorrat gewählt. Zur Optimierung eines Makroblocks ist die damit verknüpfte optimale hierarchische Struktur zu determinieren.

3.3 Optimierung von Baumstrukturen

Zur Beschreibung von Baumstrukturen verwenden wir die Definitionen aus [10]. Ein *Baum* $T = \{t_0, t_1, t_2, \ldots\}$ ist eine Menge von *Knoten* mit einer einzigen *Wurzel* t_0. Die Menge der *Endknoten* eines Baumes T ist durch $\tilde{T}$ gekennzeichnet und ist eine Teilmenge von T. Ein *Teilbaum* S des Baumes T ist ein Baum mit Wurzel $t \in T$, dessen Endknoten $\tilde{S}$ nicht notwendigerweise eine Teilmenge von $\tilde{T}$ bilden und *innere Knoten* von T enthalten können. Es gibt zwei besondere Arten von Teilbäumen von T. Wenn tatsächlich $\tilde{S} \subseteq \tilde{T}$ gilt, dann bezeichnet man diesen Teilbaum als *Zweig* von T am Knoten t und wird durch T_t gekennzeichnet. Wenn im anderen Fall $t = t_0$ gilt, dann spricht man von einem *verjüngten Teilbaum* von T und kennzeichnet ihn mit $T^p \preceq T$. Wenn beide Fälle zutreffen, dann gilt $T^p = T_t = T$. **Abbildung 3.3** veranschaulicht diese Definitionen.

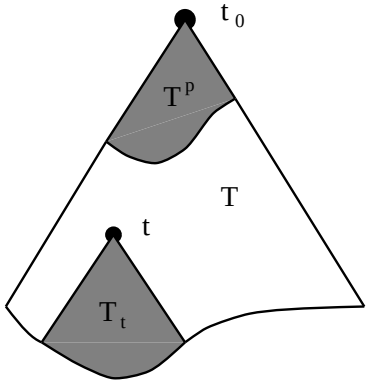


Abbildung 3.3: *Typen von Teilbäumen. T ist der gesamte Baum mit Wurzel t_0. T_t ist ein Zweig von T mit der Wurzel t. $T^p \preceq T$ ist ein verjüngter Teilbaum von T.*

Eine reellwertige Funktion über einem Baum, d.h. eine reellwertige Funktion über einer Menge von Knoten, bezeichnet man als *Baumfunktional*. Werden nun den Makroblöcken Baumstrukturen überlagert, so bezeichnet man das Lagrange-Funktional des zu codierenden Makroblocks auch als Baumfunktional.

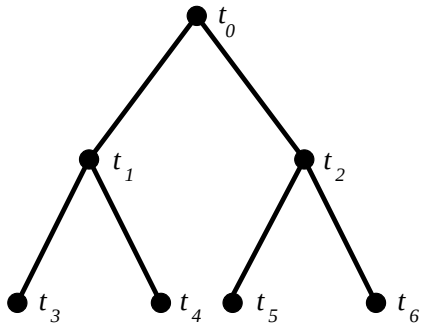


Abbildung 3.4: *Beispiel eines Baumes mit bis zu zwei Zweigen pro Knoten.*

Wir betrachten ein Baumfunktional $J(T)$ über dem Baum T in **Abbildung 3.4.**

$$J(T) = J(\{t_0, t_1, t_2, \ldots, t_6\}) \tag{3.12}$$

Besteht keine *Abhängigkeit* zwischen den Knoten, d.h. Bildpunkten des Makroblocks, so können die Knoten unabhängig voneinander betrachtet werden.

$$\begin{aligned} J(T) &= J(\{t_0\} \cup \{t_1\} \cup \{t_2\} \cup \ldots \cup \{t_6\}) && (3.13)\\ &= \sum_{i=0}^{6} J(\{t_i\}) && (3.14) \end{aligned}$$

Liegt diese Unabhängigkeit zwischen den Knoten eines Baumes vor, so bezeichnet man das Funktional als *affines Baumfunktional*.

Mit diesen Definitionen versuchen wir nun die optimale Baumstruktur zu bestimmen. Die optimale Baumstruktur oder der optimale verjüngte Teilbaum ist durch das Minimum des Baumfunktionals definiert.

Um die Komplexität des Optimierungsproblems darzustellen, betrachten wir einen Baum mit bis zu vier Zweigen pro Knoten. Die Kardinalität des Suchraumes ist identisch mit der Anzahl aller möglichen verjüngten Teilbäume.

$$\begin{aligned} C_f(d+1) &= [C_f(d)+1]^4 && (3.15)\\ C_f(d+1) &= C_f^4(d) + 4C_f^3(d) + 6C_f^2(d) + 4C_f(d) + 1 && (3.16) \end{aligned}$$

$C_f(d)$ – die Anzahl der Bäume mit bis zu vier Zweigen pro Knoten, deren Tiefe kleiner oder gleich d ist – ist durch eine rekursive Beziehung gegeben, wobei d die maximale Tiefe des Baumes bezeichnet.

$$\begin{aligned} C_f(0) &= 1\\ C_f(1) &= 16\\ C_f(2) &= 83521\\ C_f(3) &= 4.8 \cdot 10^{19} \end{aligned}$$

Sind nur vier oder keine Zweige pro Knoten zugelassen, so vereinfacht sich die rekursive Beziehung.

$$C_{rf}(d+1) = C_{rf}^4(d) + 1 \tag{3.17}$$

Liegt hingegen ein affines Baumfunktional vor, so ist die Kardinalität des Suchraumes proportional zu der Anzahl der inneren Knoten des Baumes. Die Anzahl der Knoten eines Baumes A mit vier Zweigen pro Knoten erhält man durch die Rekursion

$$A(d+1) = 4A(d) + 1. \tag{3.18}$$

oder die Beziehung

$$A(d) = \frac{1}{3}\left(4^{d+1} - 1\right). \tag{3.19}$$

Auf der untersten Ebene ist immer nur eine Art von Knoten (Endknoten) möglich. Für die inneren Knoten sind jeweils $2^4 = 16$ verschiedene Knotenformen zulässig, da bis zu vier Zweige pro Knoten zugelassen sind. Daraus ergibt sich für die Kardinalität des Suchraumes bzw. die Komplexität für dieses Optimierungsproblem

$$C_p(d) = 16A(d-1), \tag{3.20}$$

$$C_p(d) = \frac{16}{3}\left(4^d - 1\right). \tag{3.21}$$

$C_p(d)$ ist die Anzahl der Segmentformen der inneren Knoten, deren Lagrange-Kosten zu berechnen sind um die optimalen Segmentformen zu bestimmen.

$$\begin{aligned} C_p(0) &= 0 \\ C_p(1) &= 16 \\ C_p(2) &= 80 \\ C_p(3) &= 336 \end{aligned}$$

Für ein affines Baumfunktional mit nur vier Zweigen oder keinem Zweig pro Knoten ergibt sich für die Komplexität $C_{rp}(d)$.

$$C_{rp}(d) = \frac{1}{3}\left(4^d - 1\right) \tag{3.22}$$

Durch die Annahme, daß ein affines Baumfunktional vorliegt, reduziert sich die Komplexität bei der Tiefe Drei um den Faktor $1.4 \cdot 10^{17}$. Die Komplexität der vollständigen Suche erzwingt sogar eine solche Annahme um eine Lösung des Optimierungsproblems zu realisieren.

Die folgenden Algorithmen erlauben nun die Bestimmung der Baumstrukturen durch Minimierung des Baumfunktionals.

3.3.1 Pruning-Algorithmus

Der Pruning-Algorithmus erlaubt die Bestimmung der optimalen Baumstruktur $\widehat{T^p}$, wenn ein affines Baumfunktional vorliegt. In [10] wird dieser Algorithmus auch als *Mincost-Algorithmus* bezeichnet. Die optimale Baumstruktur oder den optimalen verjüngten Teilbaum erhalten wir durch Minimierung des affinen Baumfunktionals.

$$J(\widehat{T^p}) = \min_{\{T^p : T^p \preceq T\}} J(T^p) \tag{3.23}$$

Bezeichnet G den verjüngten Teilbaum von T der Tiefe Eins und setzt man die Unabhängigkeit zwischen G und den daran hängenden Zweigen T_t mit $t \in \tilde{G}$ voraus, so wird der Baum T wie folgt zerlegt:

$$T = G \backslash \tilde{G} \bigcup_{t \in \tilde{G}} T_t. \tag{3.24}$$

Es ist zu bemerken, daß die Knoten $\{t : t \in \tilde{G}\}$ sowohl in G als auch in T_t enthalten sind. Die vorausgesetzte Unabhängigkeit der Zweige T_t untereinander erlaubt die unabhängige Optimierung.

$$\min_{T^p} J(T^p) = \min_{G^p, T_t^p} J(G^p \backslash \widetilde{G^p} \bigcup_{t \in \widetilde{G^p}} T_t^p) \tag{3.25}$$

$$= \min_{G^p} \left\{ J(G^p \backslash \widetilde{G^p}) + \sum_{t \in \widetilde{G^p}} \min_{T_t^p} J(T_t^p) \right\} \tag{3.26}$$

Wir erhalten die rekursive Optimierungsvorschrift, die die optimale Baumstruktur für affine Baumfunktionale bestimmt.

$$\min_{\{T^p : T^p \preceq T\}} J(T^p) = \min_{\{G^p : G^p \preceq G\}} \left\{ J(G^p) - \sum_{t \in \widetilde{G^p}} J(\{t\}) + \sum_{t \in \widetilde{G^p}} \min_{\{T_t^p : T_t^p \preceq T_t\}} J(T_t^p) \right\} \tag{3.27}$$

Ein Beispiel soll die Optimierung der Baumstruktur in **Abbildung 3.4** verdeutlichen.

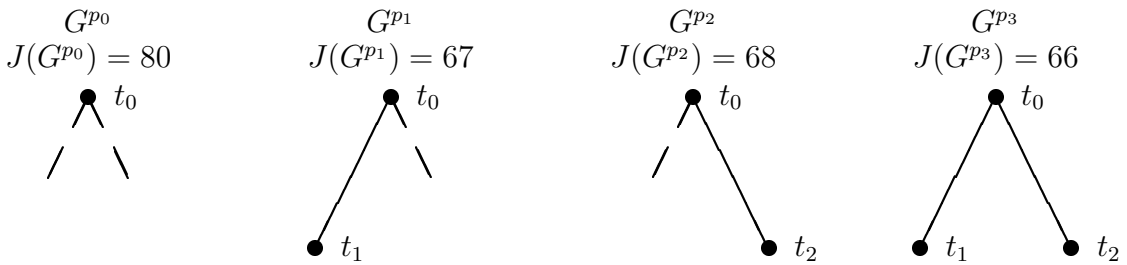


Abbildung 3.5: *Alle möglichen verjüngten Teilbäume von G und deren Kosten.*

Wir bilden die Teilbäume $G = \{t_0, t_1, t_2\}$, $T_{t_1} = \{t_1, t_3, t_4\}$ und $T_{t_2} = \{t_2, t_5, t_6\}$ und wählen die Kosten der Knoten t_1 und t_2 zu $J(\{t_1\}) = 40$ und $J(\{t_2\}) = 25$ und die optimalen Zweigkosten zu $J(\widehat{T^p_{t_1}}) = 20$ und $J(\widehat{T^p_{t_2}}) = 30$. Somit erhalten wir für die möglichen verjüngten Bäume T^p mit den Angaben aus **Abbildung 3.5**:

$$\begin{aligned}
J(T^{p_0}) &= J(G^{p_0}) = 80 \\
J(T^{p_1}) &= J(G^{p_1}) - J(\{t_1\}) + J(\widehat{T^p_{t_1}}) = 47 \\
J(T^{p_2}) &= J(G^{p_2}) - J(\{t_2\}) + J(\widehat{T^p_{t_2}}) = 72 \\
J(T^{p_3}) &= J(G^{p_3}) - J(\{t_1\}) - J(\{t_2\}) + J(\widehat{T^p_{t_1}}) + J(\widehat{T^p_{t_2}}) = 51.
\end{aligned}$$

Die minimalen Kosten ergeben sich für die Struktur G^{p_1}. Somit kann der Zweig T_{t_2} entfernt werden. Die Kosten des optimalen verjüngten Baumes $\widehat{T^p}$ ergeben sich zu $J(\widehat{T^p}) = J(T^{p_1}) = 47$. Wie wir aus diesem Beispiel erkennen, müssen die optimalen Zweigkosten $J(\widehat{T^p_{t_1}})$ und $J(\widehat{T^p_{t_2}})$ bekannt sein, bevor der optimale verjüngte Teilbaum $\widehat{T^p}$ bestimmt werden kann. Deshalb spricht man hier auch von einer *Bottom-Up-Strategy* [7].

Ist man nicht nur an einem optimalen verjüngten Teilbaum für einen einzigen Wert von λ, sondern auch an verschiedenen optimalen verjüngten Teilbäumen über einem Intervall interessiert, so ist die Methode des optimalen Prunings nach [10] aufwandsgünstiger. Beim optimalen Pruning wird zu jedem Zeitpunkt ein Knoten nach dem Rate-Distortion-Kriterium entfernt und ist so zu jedem Zeitpunkt optimal. Der Mincost-Algorithmus liefert hingegen nur die optimale endgültige Baumstruktur, die man auch nach der Methode des optimalen Prunings erhält.

Zur Schätzung der Parameter des Bewegungs- und Intensitätsmodells ist neben der Segmentform S, die die Baumstruktur festlegt, auch die örtliche Verschiebung und die Intensitätsdifferenz (Quantisierung Q) zu bestimmen. Der Pruning-Algorithmus aus **Gleichung 3.27** lautet in angepaßter Notation:

$$\hat{J}^l = \min_{S^l, Q^l} \left\{ J^l_L(S^l, Q^l) + \hat{J}^{l+1}(S^l) \right\}. \tag{3.28}$$

Der Index l kennzeichnet die Tiefe im Quadtree. Der Anteil der Kosten des aktuellen Knotens wird durch J^l_L, der Zweig-Anteil durch $\hat{J}^{l+1}$ gekennzeichnet. **Abbildung 3.6** veranschaulicht die Kostenanteile der Entscheidungsregel.

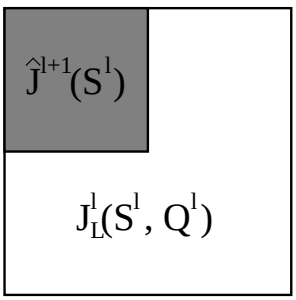


Abbildung 3.6: *Bestimmung der optimalen Segmentform und Quantisierung beim Pruning-Algorithmus. $J^l_L(S^l, Q^l)$ ist der Endknoten-Anteil der Lagrange-Kosten für einen Knoten mit der Baumtiefe l. Der Zweig-Anteil der Lagrange-Kosten $\hat{J}^{l+1}(S^l)$ ergibt sich je nach Segmentform S^l aus den Lagrange-Kosten der Zweige.*

Sind nur vier oder keine Zweige pro Knoten zugelassen, so ist für jeden Knoten nur eine binäre Entscheidung (*split, merge*) anhand der Lagrange-Kosten zu treffen. Diese vereinfachte Entscheidungsregel wird in [20], [21], [8], [9] dargestellt.

Mit der Entscheidungsregel läßt sich der Pruning-Algorithmus zur Bestimmung der Baumstruktur nach **Abbildung 3.7** formulieren. Die Operation $K := Child(K)$ setzt den aktuellen Knoten auf den nächsten hierarchisch untergeordneten Knoten; $K := Parent(K)$ setzt den aktuellen Knoten auf den hierarchisch übergeordneten Knoten.

Schritt 1: Falls die Zweig-Kosten des aktuellen Knotens K bekannt sind, fahre mit 2 fort; ansonsten setze

$$K := Child(K)$$

und kehre nach 1 zurück.

Schritt 2: Bestimme die optimale Segmentform und Quantisierung für den aktuellen Knoten nach **Gleichung 3.28**.

Schritt 3: Falls die Wurzel des Baumes erreicht ist, breche den Algorithmus ab; ansonsten setze

$$K := Parent(K)$$

und kehre nach 1 zurück.

Abbildung 3.7: *Pruning-Algorithmus zur Bestimmung der Baumstruktur*

Zur Herleitung des Pruning-Algorithmus wurde in **Gleichung 3.24** die Unabhängigkeit benachbarter Zweige (Teilblöcke) angenommen. Mit dieser Annahme erhielten wir einen optimalen Algorithmus. Da dies aber eine Vereinfachung der tatsächlichen Verhältnisse widerspiegelt (zwischen den bewegungs- und intensitätskompensierten Blöcken besteht eine örtliche Abhängigkeit), ist der Pruning-Algorithmus für diese Anwendung suboptimal.

3.3.2 Growing-Algorithmus

Der Growing-Algorithmus stellt eine weitere Möglichkeit dar, Baumstrukturen zu determinieren. Er ist eine vereinfachte Version des Pruning-Algorithmus. Der entscheidende Punkt für die Komplexitätsreduktion liegt in der Tatsache, daß nicht alle Knoten des Baumes untersucht werden. Diese Entscheidung ist abhängig von den tatsächlich vorliegenden Daten. Im folgenden zeigen wir, welche Vereinfachung wir am Pruning-Algorithmus vorzunehmen haben, um den Growing-Algorithmus zu erhalten, und wann diese Vereinfachung zu vernachlässigen ist.

In der rekursiven Optimierungsvorschrift für affine Baumfunktionale nach **Gleichung 3.27** subtrahieren sich die minimalen Zweigkosten mit den Kosten der Wurzel der Zweige. Ist nun die Summe der Differenzen gegenüber den Kosten des verjüngten Teilbaums der Tiefe Eins zu vernachlässigen

$$\sum_{t \in \widetilde{G^p}} \left\{ \min_{\{T_t^p : T_t^p \preceq T_t\}} J(T_t^p) - J(\{t\}) \right\} \ll J(G^p), \tag{3.29}$$

so vereinfacht sich der rekursive Schritt des Mincost-Algorithmus.

$$\min_{\{T^p : T^p \preceq T\}} J(T^p) \approx \min_{\{G^p : G^p \preceq G\}} J(G^p) \tag{3.30}$$

Greifen wir auf das Beispiel in **Abbildung 3.5** zurück. Für die Bestimmung der suboptimalen Struktur von G benötigen wir nicht die Kosten der Zweige T_{t_1} und T_{t_2}.

$$\min_{\{G^p : G^p \preceq G\}} J(G^p) = J(G^{p_3}) = 66 \tag{3.31}$$

Der Growing-Algorithmus entscheidet sich für die Struktur G^{p_3} im Vergleich zum Pruning-Algorithmus, der G^{p_1} wählt.

Beginnend mit der Wurzel des Baumes ist für jeden neu entstandenen Zweig der Tiefe Null **Gleichung 3.30** zu berechnen. Wird nun für einen inneren Knoten nach **Gleichung 3.30** auf einen Endknoten entschieden, so werden die Zweige dieses Knotens nicht weiter berücksichtigt. Man spricht hier von einer *Top-Down-Strategy* [7].

Dieser Algorithmus hat nur zum Ziel, die endgültige suboptimale Baumstruktur zu bestimmen. Welche Knoten in welcher Reihenfolge geprüft werden, wird nicht näher festgelegt. Der in [11] vorgeschlagene Growing-Algorithmus für den Entwurf von Vektor-Quantisierern variabler Rate (**R**iskin **G**ray **A**lgorithm) strebt hingegen dieses Ziel an. Der RGA wählt die Knoten zu jedem Zeitpunkt nach dem Rate-Distortion-Kriterium optimal. Da wir uns hier primär für einen Punkt und nicht für ein Intervall der Rate-Distortion-Kurve interessieren, legen wir mit diesem Algorithmus nicht die Reihenfolge der zu teilenden Knoten fest. (Variationen des RGA sind in [22] zu finden.)

Zur Schätzung der Parameter des Bewegungs- und Intensitätsmodells ist neben der Segmentform S, die die Baumstruktur festlegt, auch die örtliche Verschiebung und die Intensitätsdifferenz (Quantisierung Q) zu bestimmen. Der Growing-Algorithmus aus **Gleichung 3.30** lautet in angepaßter Notation:

$$\hat{J}^l = \min_{S^l, Q^l} \left\{ J_L^l(S^l, Q^l) + \hat{J}_L^{l+1}(S^l) \right\} \tag{3.32}$$

Der Index l kennzeichnet die Tiefe im Quadtree. Der Anteil der Kosten des aktuellen Knotens wird durch J_L^l, der Anteil des Zweiges der Tiefe Null durch $\hat{J}_L^{l+1}$ gekennzeichnet. **Abbildung 3.8** veranschaulicht die Kostenanteile.

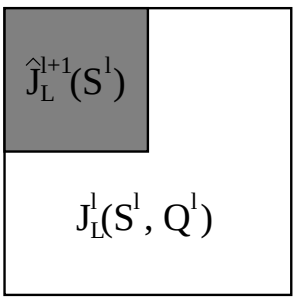


Abbildung 3.8: *Bestimmung der optimalen Segmentform und Quantisierung beim Growing-Algorithmus. $J_L^l(S^l, Q^l)$ ist der Endknoten-Anteil der Lagrange-Kosten für einen Knoten mit der Baumtiefe l. Der Zweig-Anteil der Lagrange-Kosten $\hat{J}_L^{l+1}(S^l)$ ergibt sich je nach Segmentform S^l aus den Lagrange-Kosten der Zweige der Tiefe Null.*

Mit dieser Entscheidungsregel läßt sich der Growing-Algorithmus zur Bestimmung der Baumstruktur nach **Abbildung 3.9** formulieren. Die Operation $K := Child(K)$ setzt den aktuellen Knoten auf den nächsten hierarchisch untergeordneten Knoten.

Schritt 0: Beginne mit der Wurzel des Baumes.

Schritt 1: Bestimme die Zweig-Kosten des aktuellen Knotens K, indem man für jeden Zweig einen Baum der Tiefe Null annimmt.

Schritt 2: Bestimme die optimale Segmentform und Quantisierung für den aktuellen Knoten nach **Gleichung 3.32**.

Schritt 3: Für jeden Zweig, der den aktuellen Knoten K verläßt, setze

$$K := Child(K)$$

und kehre nach 1 zurück. Sind keine Zweige mehr vorhanden, so ist der Algorithmus beendet.

Abbildung 3.9: *Growing-Algorithmus zur Bestimmung der Baumstruktur*

Der Growing-Algorithmus stützt sich wie der Pruning-Algorithmus auf die Annahme der Unabhängigkeit benachbarter Zweige (Teilblöcke). Da aber zwischen den bewegungs- und intensitätskompensierten Blöcken eine örtliche Abhängigkeit besteht und die Vereinfachung nach **Gleichung 3.29** zugelassen wird, ist der Growing-Algorithmus für diese Anwendung in zweifacher Hinsicht suboptimal.

3.4 Simulationsergebnisse

Für den Vergleich zwischen Pruning- und Growing-Algorithmus geben wir ein optimales Codebuch nach **Kapitel 4** vor. Die Optimierung ist mit dem Pruning-Algorithmus bei $\lambda = 150$ an den Trainingssequenzen durchgeführt. Bewegungs- und Intensitätskompensation ist für die Y-Farbkomponente mit Blöcken der Größe 16×16, 8×8, 4×4 und 2×2 bzw. für die U- und V-Farbkomponenten mit Blöcken der Größe 8×8, 4×4 und 2×2 realisiert. Zusätzlich sind die Blöcke variabler Größe in sechzehn Segmente zerlegt. Die Simulationsbedingungen für die folgenden Ergebnisse sind in **Anhang C** erläutert.

3.4.1 Rate-Distortion-Verhalten

Abbildungen 3.10 und 3.11 zeigen das Rate-Distortion-Verhalten des Pruning- bzw. Growing-Algorithmus für die Testsequenzen in QCIF-Auflösung und **Abbildungen 3.12 und 3.13** das der Testsequenzen in CIF-Auflösung. Die Testsequenzen mit jeweils einer Dauer von 10 Sekunden werden mit 7.5 Bildern pro Sekunde abgetastet. Dabei wurden alle Testsequenzen für verschiedene λ-Werte codiert.

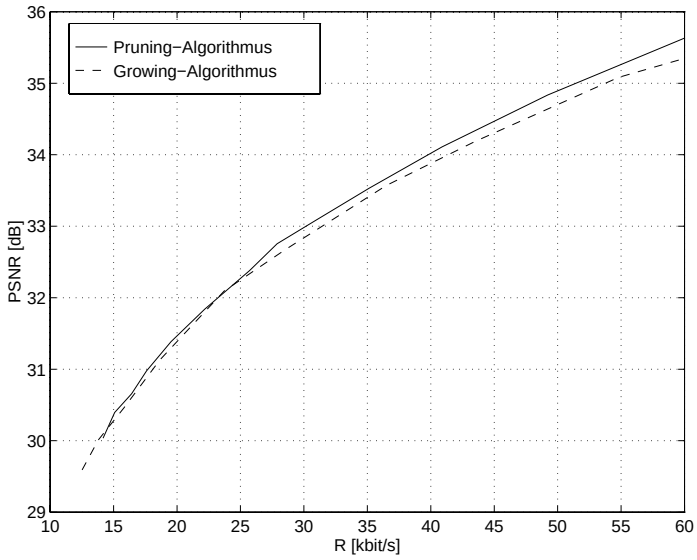


Abbildung 3.10: *Vergleich zwischen Pruning- und Growing-Algorithmus bei vorgegebenem Codebuch anhand der Testsequenz „Car Phone“ (QCIF, 7.5 fps, 10 s).*

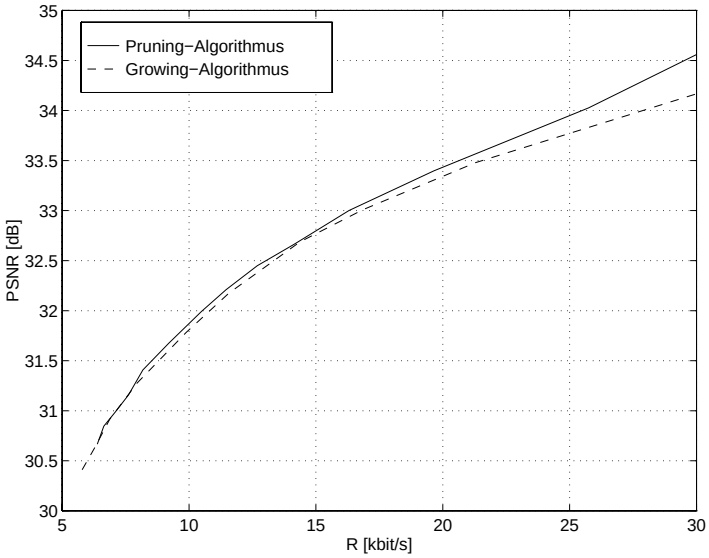


Abbildung 3.11: *Vergleich zwischen Pruning- und Growing-Algorithmus bei vorgegebenem Codebuch anhand der Testsequenz „Salesman“ (QCIF, 7.5 fps, 10 s).*

Man erkennt aus den **Abbildungen 3.10, 3.11, 3.12 und 3.13**, daß sich die beiden Algorithmen bei niedrigen Raten ähnlich verhalten. Die entstandenen Baumstrukturen ähneln sich in dem Sinne, daß beide nahezu gleiche Lagrange-Kosten aufweisen. Daraus läßt sich folgern, daß die Näherung nach **Gleichung 3.29** zutrifft. Die Summe der Differenzen der Lagrange-Kosten der Zweige T_t und der der Wurzeln $\{t\}$ sind gegenüber denjenigen des verjüngten Teilbaums der Tiefe Eins G zu vernachlässigen.

Betrachten wir einen Knoten t für den die Lagrange-Kosten des Zweiges T_t und die der Wurzel $\{t\}$ näherungsweise gleich sind.

$$J(\{t\}) \approx \min_{\{T_t^p : T_t^p \preceq T_t\}} J(T_t^p) \tag{3.33}$$

Näherungsweise gleiche Lagrange-Kosten implizieren per Definition näherungsweise gleiche Baumstrukturen. Somit gleicht der optimale verjüngte Teilbaum $\widehat{T_t^p} \preceq T_t$ näherungsweise der Wurzel $\{t\}$.

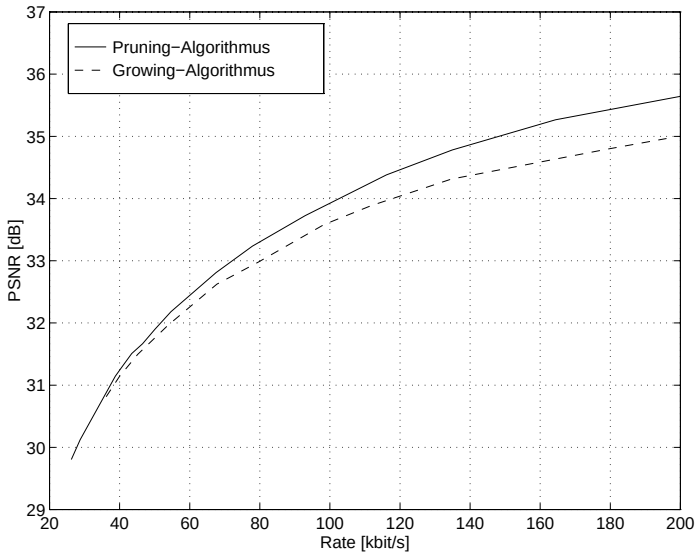


Abbildung 3.12: *Vergleich zwischen Pruning- und Growing-Algorithmus bei vorgegebenem Codebuch anhand der Testsequenz „Car Phone“ (CIF, 7.5 fps, 10 s).*

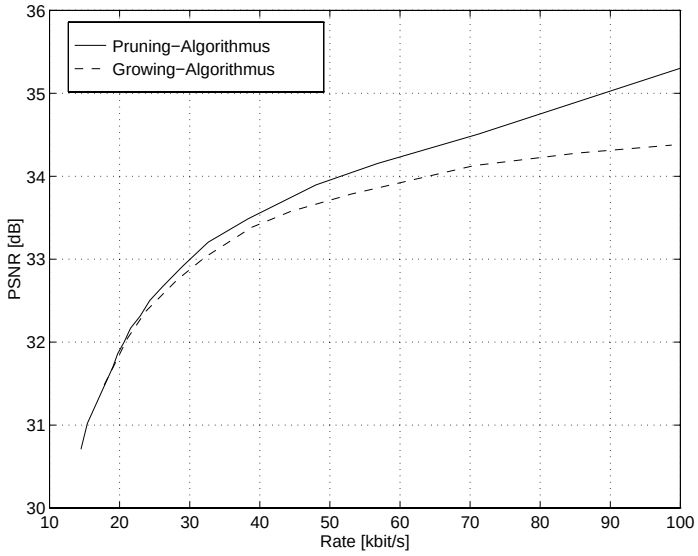


Abbildung 3.13: *Vergleich zwischen Pruning- und Growing-Algorithmus bei vorgegebenem Codebuch anhand der Testsequenz „Salesman“ (CIF, 7.5 fps, 10 s).*

$$\{t\} \approx \widehat{T_t^p} \preceq T_t \tag{3.34}$$

Bei zunehmender Anzahl der Knoten, die durch ihre Wurzel $\{t\}$ repräsentiert werden, werden die optimal verjüngten Baumstrukturen „flacher“, d.h. die mittlere Baumtiefe verkleinert sich bei sinkender Rate der Videosequenz.

Somit läßt sich schließen, daß für abnehmende Datenrate überwiegend größere Segmente (Blöcke) für die Bewegungs- und Intensitätskompensation herangezogen werden.

3.4.2 Recheneffizienz

Zur Beurteilung der Recheneffizienz der Algorithmen sind in den **Tabellen 3.1, 3.2, 3.3 und 3.4** die Codierergebnisse der Testsequenzen dargestellt. Die oben erläuterten Simulationsbedingungen sind auch hier erfüllt.

Algorithmus	Pruning	Growing
$R[\frac{kbit}{s}]$	24.0	24.0
$PSNR[dB]$	32.2	32.2
λ	210	200
$T_{Encode}[\frac{s}{Bild}]$	16.8	11.3
T_{rel}	1.00	0.67

Tabelle 3.1: *Beurteilung der Recheneffizienz der Algorithmen anhand der Testsequenz „Car Phone“ (QCIF, 7.5 fps, 10 s) (SUN UltraSparc1).*

Algorithmus	Pruning	Growing
$R\ [\frac{kbit}{s}]$	10.0	10.0
$PSNR\ [dB]$	31.9	31.8
λ	200	190
$T_{Encode}\ [\frac{s}{Bild}]$	16.6	8.90
T_{rel}	1.00	0.54

Tabelle 3.2: *Beurteilung der Recheneffizienz der Algorithmen anhand der Testsequenz „Salesman“ (QCIF, 7.5 fps, 10 s) (SUN UltraSparc1).*

Algorithmus	Pruning	Growing
$R\ [\frac{kbit}{s}]$	48.0	48.0
$PSNR\ [dB]$	31.7	31.6
λ	440	360
$T_{Encode}\ [\frac{s}{Bild}]$	67.1	50.6
T_{rel}	1.00	0.75

Tabelle 3.3: *Beurteilung der Recheneffizienz der Algorithmen anhand der Testsequenz „Car Phone“ (CIF, 7.5 fps, 10 s) (SUN UltraSparc1).*

Algorithmus	Pruning	Growing
$R\ [\frac{kbit}{s}]$	20.0	20.0
$PSNR\ [dB]$	31.9	31.9
λ	540	460
$T_{Encode}\ [\frac{s}{Bild}]$	48.2	30.8
T_{rel}	1.00	0.64

Tabelle 3.4: *Beurteilung der Recheneffizienz der Algorithmen anhand der Testsequenz „Salesman“ (CIF, 7.5 fps, 10 s) (SUN UltraSparc1).*

Die Codierung der Testsequenzen in QCIF-Auflösung bei 24 $\frac{kbit}{s}$ bzw. bei 10 $\frac{kbit}{s}$ mit dem Growing-Algorithmus weist eine Codierzeitersparnis von 33% bzw. 46% gegenüber dem Pruning-Algorithmus auf. Die Codierung der Testsequenzen in CIF-Auflösung bei 48 $\frac{kbit}{s}$ bzw. bei 20 $\frac{kbit}{s}$ mit dem Growing-Algorithmus weist eine Codierzeitersparnis von 25% bzw. 36% gegenüber dem Pruning-Algorithmus auf. Bei diesen Raten ist die Distortion für beide Algorithmen annähernd gleich. Darüber hinaus ist festzustellen, daß die Codierzeitersparnis vom Sequenzinhalt abhängig ist.

Kapitel 4

Quantisierer-Entwurf mit Raten-Nebenbedingung

4.1 Einführung

Der Video-Codec läßt sich in zwei Hauptbestandteile zerlegen: die Modellierung der Videosequenzen durch das Bewegungs- und Intensitätsmodell (Markov-Quelle erster Ordnung) und Quantisierung der Parametersequenzen. Beschreiben wir die Videoquelle mit der Zufallsvariablen $\mathbf{X}$, so bildet das Bewegungs- und Intensitätsmodell (MIM) diese auf das Modell $\mathbf{U}$ ab. Diese Abbildung reduziert vor allem die zeitliche Korrelation des stochastischen Prozesses $\mathbf{X}(t)$. Das Modell $\mathbf{U}$ bilden wir auf das quantisierte Modell $\mathbf{V}$ ab und führen dabei die Redundanzreduktion der Quelle durch. Das MIM erlaubt die Rekonstruktion zur Videosenke $\mathbf{Y}$ mit dem quantisierten Modell $\mathbf{V}$.

Der **G**eneralized **L**loyd **A**lgorithm (GLA) [16] erlaubt den optimalen Quantisierer-Entwurf für das Modell. Für das Codierproblem sind wir aber zusätzlich an der Minimierung der mittleren Information (Entropie), die nötig ist um Videosequenzen zu rekonstruieren, interessiert. Diese Quantisierung mit Entropie-Nebenbedingung ermöglicht der iterative Algorithmus nach **C**hou, **L**ookabaugh und **G**ray (CLGA) [15]. (Siehe auch *Weighted Universal Vector Quantization* [23].)

In diesem Kapitel erläutern wir den Entwurf des Quantisierers mit Raten-Nebenbedingung für die Parameter des MIM und verdeutlichen die Ergebnisse an Simulationen.

4.2 Parameter des MIM

Die quantisierten Parameter des MIM beschreiben wir durch die mehrdimensionale Zufallsvariable $\mathbf{V}$ (quantisiertes Modell), die sich aus der örtlichen Verschiebung und der Intensitätsdifferenz zusammensetzt.

$$\mathbf{v} = (\boldsymbol{\Delta}\mathbf{m}, \boldsymbol{\Delta}\mathbf{n}, \mathbf{q}) \tag{4.1}$$

Für die Videosenke $\mathbf{Y}$ nehmen wir ein Markov-Modell erster Ordnung an. Das Bild $\mathbf{y}$[k] zur diskreten Zeit k erhalten wir durch Bewegungs- und Intensitätskompensation aus dem Bild $\mathbf{y}$[k-1] zur diskreten Zeit $k-1$.

$$\mathbf{y}[m,n,k] = \mathbf{y}[m+\mathbf{\Delta m}, n+\mathbf{\Delta n}, k-1] + \mathbf{q} \tag{4.2}$$

Abbildung 4.1 visualisiert das Zeitverhalten des Bewegungs- und Intensitätsmodells. Das Modell $\mathbf{U}$ wird durch Quantisierung auf das quantisierte Modell $\mathbf{V}$ abgebildet.

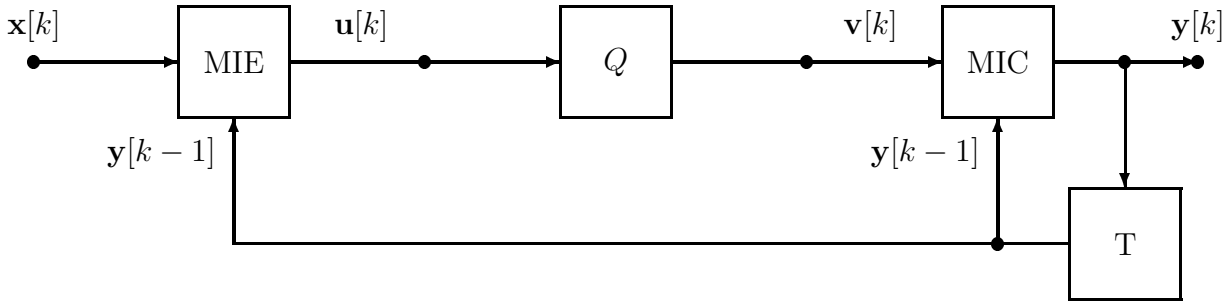


Abbildung 4.1: *Zeitverhalten des MIM*

Für das Modell definieren wir die Distortion D mit Hilfe des konvexen Fehlermaßes ρ. Wir wählen den quadratischen Fehler zwischen einem Original $\mathbf{x}$ und einer mit dem Modell generierten Rekonstruktion $\mathbf{y}$.

$$\rho(\mathbf{u},\mathbf{v}) = \sum_{m,n,k} |\mathbf{x}[m,n,k] - \mathbf{y}[m,n,k]|^2 \tag{4.3}$$

Die Distortion zwischen tatsächlichem und quantisiertem Modell erhalten wir durch Bildung des Erwartungswertes.

$$D(\mathbf{U},\mathbf{V}) = E\left\{\rho(\mathbf{U},\mathbf{V})\right\} \tag{4.4}$$

Mit diesen Überlegungen zum MIM können wir uns dem iterativen Algorithmus für den Entwurf des MIM-Parameter-Quantisierers zuwenden.

4.3 Entwurf des MIM-Parameter-Quantisierers

Berücksichtigen wir die Raten-Nebenbedingung nicht, so löst der GLA iterativ die nichtlineare Programmierung. Voraussetzung dabei ist die Konvexität des Fehlermaßes ρ und eine stetige Verteilungsfunktion. Das Konvergenz-Theorem nach [24] zeigt, daß die Sequenz der Quantisierer gegen den Fixpunkt-Quantisierer strebt. Dabei ist der Fixpunkt-Quantisierer derjenige, der das Problem optimal löst.

Die mathematische Formulierung für das Optimierungsproblem lautet

$$\min_{Q} E\left\{\rho(\mathbf{U}, Q(\mathbf{U}))\right\}. \tag{4.5}$$

Die optimale Abbildung Q, die die **Optimierungsvorschrift 4.5** löst, ist diejenige, die das Argument $\rho(\mathbf{u}, Q(\mathbf{u}))$ fast überall minimiert, d.h. der optimale Quantisierer Q bildet $\mathbf{u}$ auf $\mathbf{v}$ ab, wenn das Fehlermaß $\rho(\mathbf{u}, \mathbf{v})$ minimal wird [15].

$$Q(\mathbf{u}) = \underset{\mathbf{v}\in\mathbf{V}}{\operatorname{argmin}}\, \rho(\mathbf{u}, \mathbf{v}) \tag{4.6}$$

Für die Berücksichtigung der Raten-Nebenbedingung nehmen wir für die Quantisierung das Modell nach **Abbildung 4.2** an [15]. Die Abbildung α bildet das tatsächliche Modell $\mathbf{U}$ auf den Index $\mathbf{I}$ ab und ist im allgemeinen mit einem Informationsverlust verbunden. Der Operator γ weist dem Index eineindeutig den präfix-freien Code zu. Die Abbildung $\gamma \circ \alpha$ im Coder weist somit dem Modell $\mathbf{U}$ dem Code $\mathbf{C}$ zu. Wegen der eineindeutigen Zuordnung γ ist nun der Decoder in der Lage, aus dem Code den Index zu erzeugen. β bildet schließlich auf das quantisierte Modell $\mathbf{V}$ ab.

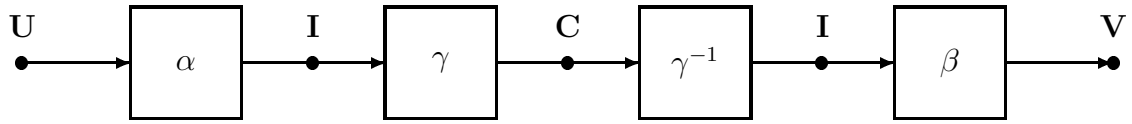


Abbildung 4.2: *Quantisierungsmodell*

Die Optimierung mit Raten-Nebenbedingung lösen wir mit Hilfe des Lagrange-Funktionals.

$$J(\alpha, \beta, \gamma, \lambda, \mathbf{U}) = E\left\{\rho(\mathbf{U}, \beta \circ \alpha(\mathbf{U})) + \lambda|\gamma \circ \alpha(\mathbf{U})|\right\} \tag{4.7}$$

Für die folgende Erklärung des iterativen Algorithmus ist $\lambda = \lambda_c$ konstant gewählt.

1. Im ersten Teilschritt bestimmen wir die optimale Abbildung α analog zu **Vorschrift 4.5 und Gleichung 4.6**. Dazu nehmen wir sowohl die Abbildungen β und γ als auch die Verteilung des Modells $F_{\mathbf{U}}(\mathbf{u})$ als konstant an.

$$\min_{\alpha} E\left\{\rho(\mathbf{U}, \beta_c \circ \alpha(\mathbf{U})) + \lambda_c|\gamma_c \circ \alpha(\mathbf{U})|\right\} \tag{4.8}$$

$$\Longrightarrow \quad \alpha(\mathbf{u}) = \underset{\mathbf{i}\in\mathbf{I}}{\operatorname{argmin}}\, \left\{\rho(\mathbf{u}, \beta_c(\mathbf{i})) + \lambda_c|\gamma_c(\mathbf{i})|\right\} \tag{4.9}$$

   Gleichung 4.9 beschreibt die Vorschrift zur optimalen Bitverteilung wie sie in **Kapitel 3** ausführlich beschrieben wird.

2. Im zweiten Teilschritt bestimmen wir die optimale Abbildung γ. Dazu nehmen wir die Abbildung α und β als konstant an. Bei konstantem α und Verteilung des Modells $F_{\mathbf{U}}(\mathbf{u})$ erhalten wir eine ebenfalls konstante Verteilung des Index $F_{\mathbf{I}}(\mathbf{i})$.

$$\begin{aligned} & \min_{\gamma} E\left\{\rho(\mathbf{U}, \beta_c \circ \alpha_c(\mathbf{U})) + \lambda_c|\gamma \circ \alpha_c(\mathbf{U})|\right\} \\ \Longrightarrow \quad & \min_{\gamma} E\left\{|\gamma \circ \alpha_c(\mathbf{U})|\right\} \\ \Longrightarrow \quad & \min_{\gamma} E\left\{|\gamma(\mathbf{I})|\right\} \end{aligned} \tag{4.10}$$

   Gleichung 4.10 besagt, daß der optimale präfix-freie Code eine minimale mittlere Codewortlänge $\sum_{\mathbf{i}\in\mathbf{I}} |\gamma(\mathbf{i})| f_{\mathbf{I}}(\mathbf{i})$ aufzuweisen hat. Für eine endliche Anzahl von Indizes

wird das Minimierungsproblem mit dem Huffman-Algorithmus gelöst. (Siehe **Abschnitt B.1.2**.) Die mit dem Huffman-Algorithmus erzielte mittlere Codewortlänge kann durch die Indexentropie abgeschätzt werden [13].

$$H(\mathbf{I}) \leq E\left\{|\gamma(\mathbf{I})|\right\} \leq H(\mathbf{I}) + 1 \tag{4.11}$$

Diese Teilschritte werden nun nach einer Initialisierung mit $(\alpha^0, \beta^0, \gamma^0)$ solange wiederholt, bis der Fixpunkt-Quantisierer $(\alpha^\kappa, \beta^0, \gamma^\kappa)$ mit $\kappa \to \infty$ bestimmt ist. In der Praxis kann davon ausgegangen werden, daß man sich bei einer relativen Änderung der Lagrange-Kosten von weniger als $\epsilon = 0.5\%$ dem Fixpunkt-Quantisierer bereits sehr gut angenähert hat [15].

$$\frac{J^{\kappa-1} - J^\kappa}{J^{\kappa-1}} \leq \epsilon \tag{4.12}$$

Der oben beschriebene Algorithmus verbessert in keinem Iterationsschritt die initiale Abbildung β^0. In dieser Form wird auch der Algorithmus für die folgenden Untersuchungen angewendet. Soll dies aber in einer Erweiterung zusätzlich durchgeführt werden, so ist in einem 3. Teilschritt bei konstantem α, γ und konstanter Verteilung des Index die optimale Abbildung β für jede Indexsequenz zu bestimmen.

$$\begin{aligned}
& \min_\beta E\left\{\rho(\mathbf{U}, \beta \circ \alpha_c(\mathbf{U})) + \lambda_c |\gamma_c \circ \alpha_c(\mathbf{U})|\right\} \\
\Longrightarrow \quad & \min_\beta E\left\{\rho(\mathbf{U}, \beta \circ \alpha_c(\mathbf{U}))\right\} \\
\Longrightarrow \quad & \min_\beta \sum_{\mathbf{i} \in \mathbf{I}} E\left\{\rho(\mathbf{U}, \beta(\mathbf{i}))\middle|\alpha_c(\mathbf{U}) = \mathbf{i}\right\} f_{\mathbf{I}}(\mathbf{i}) \\
\Longrightarrow \quad & \beta(\mathbf{i}) = \operatorname*{argmin}_{\mathbf{v} \in \mathbf{V}} E\left\{\rho(\mathbf{U}, \mathbf{v})\middle|\alpha_c(\mathbf{U}) = \mathbf{i}\right\}
\end{aligned} \tag{4.13}$$

Gleichung 4.13 bestimmt die optimale Abbildung β, d.h. den besten *Zentroiden* $\beta(\mathbf{i})$ der *Voronoi-Region* $\left\{\mathbf{U}\middle|\alpha_c(\mathbf{U}) = \mathbf{i}\right\}$. Problematisch ist hier die Tatsache, daß das quantisierte Modell nur diskrete Werte annimmt und nur eine vollständige Suche die optimale Abbildung β bestimmt.

In der vorliegenden Arbeit sind für die Zentroiden Gitterpunkte gewählt. Für die örtlichen Verschiebungen ist ein zweidimensionales *Integer-Lattice* mit Abstand Eins, für die Intensitätsdifferenzen ein eindimensionales *Integer-Lattice* mit Abstand Vier gewählt. Bei einer geschätzen Häufigkeit eines Clusters der örtlichen Verschiebung von Null wird der Zentroid aus dem Gitter entfernt.

4.3.1 Schätzung der Verteilung

Der GLA und CLGA optimieren sowohl bekannte Verteilungen als auch unbekannte, die durch eine Trainingsmenge approximiert werden. In [24] wird gezeigt, daß für wachsende Kardinalität der Trainingsmenge

1. der Fixpunkt-Quantisierer der Trainingsmenge gegen den Fixpunkt-Quantisierer der wahren Verteilung konvergiert und

2. die konvergierenden Quantisierer des Algorithmus mit der Verteilung der Trainingsmenge sich nicht schlechter verhalten als die konvergierenden Quantisierer des Algorithmus mit der wahren Verteilung.

Aufgrund dieser Tatsache schätzen wir die Verteilung des Modells, d.h. der Einträge im Codebuch, basierend auf einer Trainingsmenge von 18 Videosequenzen von je zehn Sekunden: *Akiyo, Bream, Children, Coastguard, Container Ship, Foreman, Fun Fair, Hall Monitor, Mobile, Mother and Daughter, News, Sean, Silent, Stefan, Table Tennis, Total Destruction, Tunnel* und *Weather*.

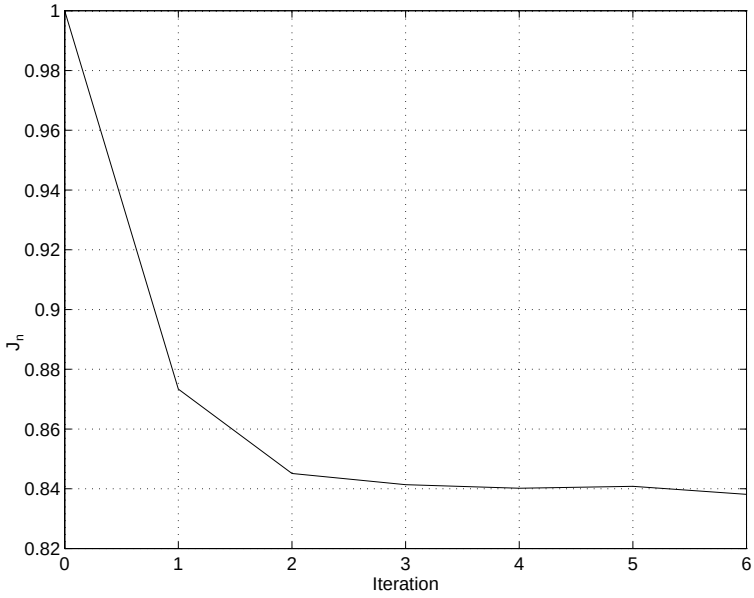


Abbildung 4.3: *Konvergenz des iterativen Algorithmus für den Codebuch-Entwurf mit $\lambda = 150$. Die Lagrange-Kosten aller Trainingssequenzen (QCIF, 7.5 fps, 10 s) sind normiert aufgetragen.*

Abbildung 4.3 zeigt die Konvergenz des iterativen Algorithmus für den Codebuch-Entwurf mit $\lambda = 150$. Die Verteilung wird aufgrund der Trainingsmenge in QCIF-Auflösung und 7.5 Bildern pro Sekunde geschätzt. Die Lagrange-Kosten der Trainingsmenge werden nach **Gleichung 4.14** normiert aufgetragen.

$$J_n^\kappa = \frac{J^\kappa}{J^0} \tag{4.14}$$

Voraussetzung für die Konvergenz ist die asymtotische Stationarität des Mittelwertes des Modells **U** [15]. Durch das Ergebnis in **Abbildung 4.3** kann diese Annahme als erfüllt betrachtet werden. Somit sind die theoretischen Überlegungen im Rahmen der Simulationsgenauigkeit anhand der vorliegenden Simulationsergebnisse verifizierbar.

Abbildung 4.4 zeigt die geschätzten Wahrscheinlichkeitsdichtefunktionen der örtlichen Verschiebungen der Y-Komponente bzw. **Abbildung 4.5** die der Intensitätsdifferenz der Y-Komponente für das Quasi-Fixpunkt-Codebuch bei $\lambda = 150$. Zu bemerken ist, daß für zunehmende Baumtiefe die Wahrscheinlichkeitsdichtefunktionen „abflachen“. Die Intensitätsdifferenz der obersten Ebene beträgt immer Null. Die wahre Wahrscheinlichkeitsdichtefunktion ist somit ein Dirac-Impuls bei Null und ist in **Abbildung 4.5** *nicht* eingezeichnet. Weiterhin weisen die Wahrscheinlichkeitsdichtefunktionen der örtlichen Verschiebungen einen Vorzug in x-Richtung auf. Dies ist möglicherweise auf das Seitenverhältnis des Bildformates 176×144 zurückzuführen.

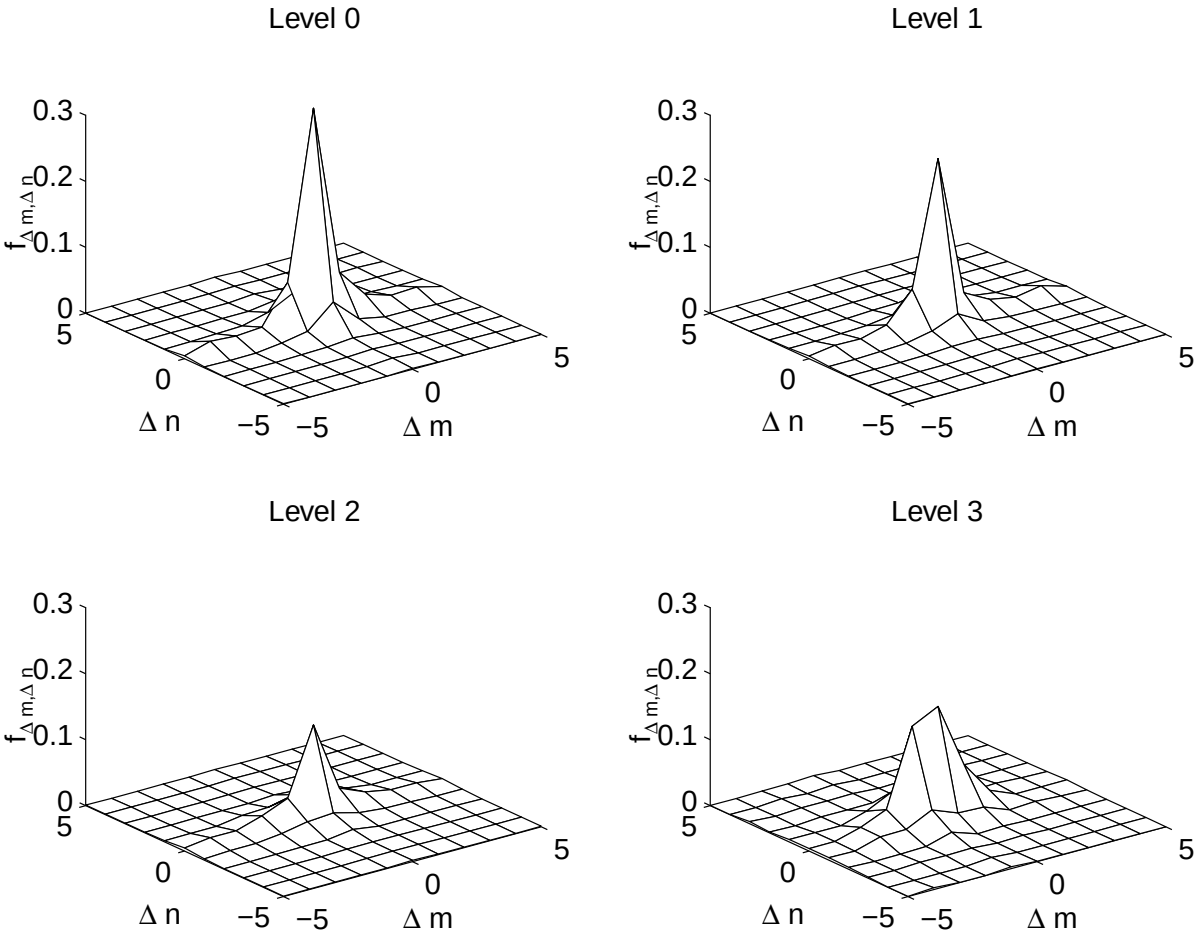


Abbildung 4.4: *Geschätzte Wahrscheinlichkeitsdichtefunktionen der örtlichen Verschiebungen für den Codebuch-Entwurf mit* $\lambda = 150$.

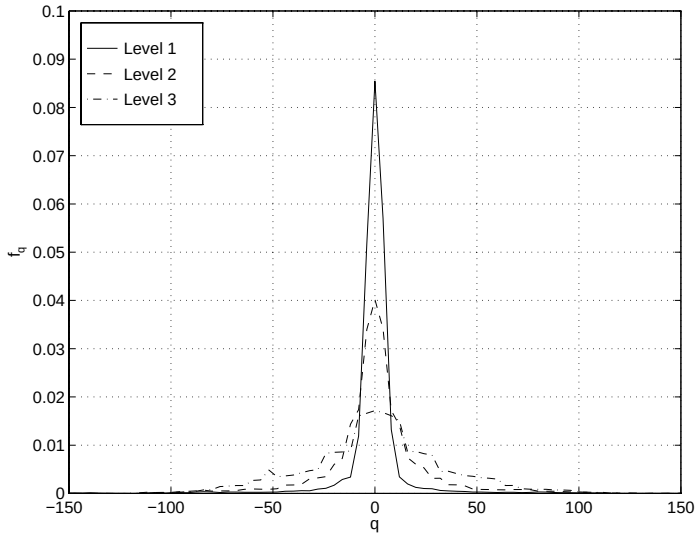


Abbildung 4.5: *Geschätzte Wahrscheinlichkeitsdichtefunktionen der Intensitätsdifferenzen für den Codebuch-Entwurf mit* $\lambda = 150$.

4.3.2 Codierergebnisse

Abbildung 4.6 und 4.7 verdeutlichen die Folge der Codebücher für $\lambda = 150$ anhand der codierten Testsequenzen „Car Phone“ und „Salesman“ in QCIF-Auflösung und 7.5 fps. Die Testsequenzen sind bei konstantem Codebuch dabei mit mehreren λ-Werten codiert. Das initiale Codebuch ist gleichverteilt, d.h. alle Zentroiden der Voronoi-Regionen sind gleichwahrscheinlich; man spricht auch von einem *Fixed Length Code*. Die Rate-Distortion-Kurven sind nur im Punkt mit der Steigung $\lambda = 150$ optimal. Die Optimalität in der Umgebung von $\lambda = 150$ wird im nächsten Abschnitt untersucht.

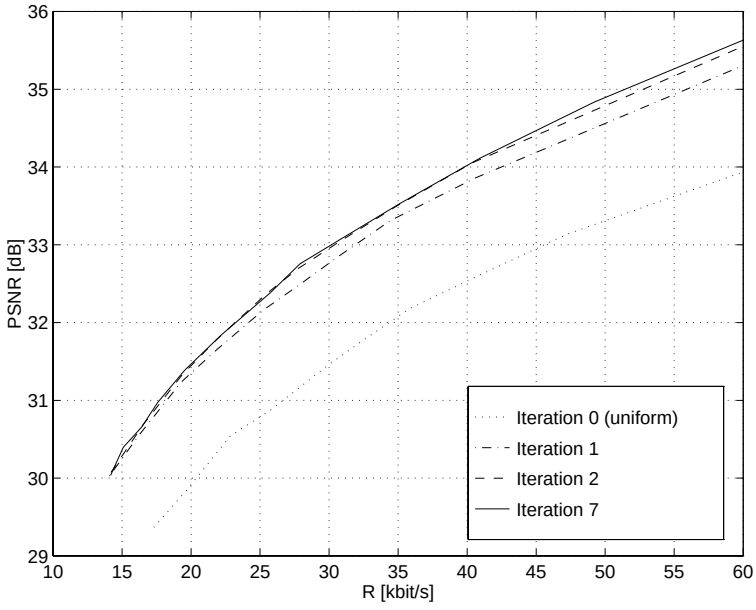


Abbildung 4.6: *Konvergenz des iterativen Algorithmus für den Codebuch-Entwurf mit $\lambda = 150$. Die Testsequenz „Car Phone“ (QCIF, 7.5 fps, 10 s) wurde mit verschiedenen λ-Werten codiert.*

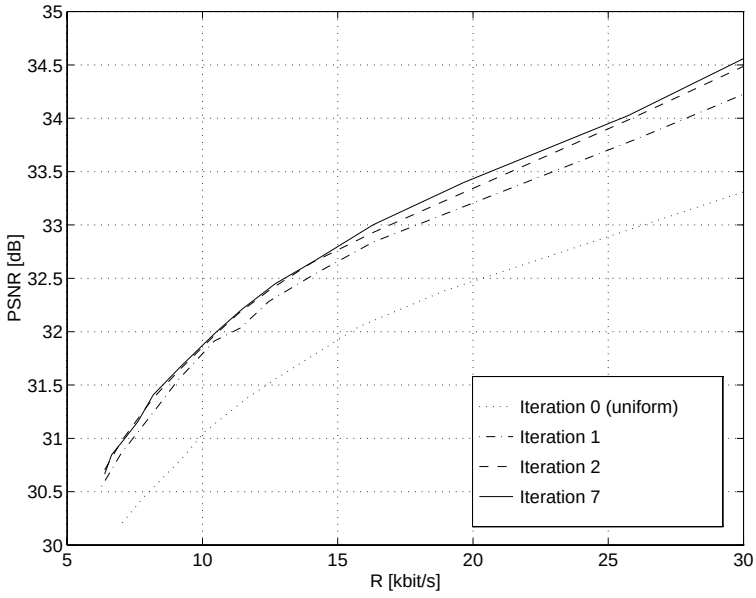


Abbildung 4.7: *Konvergenz des iterativen Algorithmus für den Codebuch-Entwurf mit $\lambda = 150$. Die Testsequenz „Salesman“ (QCIF, 7.5 fps, 10 s) wurde mit verschiedenen λ-Werten codiert.*

Das Rate-Distortion-Verhalten des Quasi-Fixpunkt-Codebuchs hat sich gegenüber dem des initialen Codebuchs bei der QCIF-Testsequenz „Car Phone“ um ca. 1.5 dB und bei der QCIF-Testsequenz „Salesman“ um ca. 1 dB verbessert.

Die Trainingssequenzen in CIF-Auflösung und 7.5 Bildern pro Sekunde stellen eine zweite Trainingsmenge dar, mit der der iterative Algorithmus verifiziert werden kann. Für die Trainingsmenge in CIF-Auflösung ist ausgehend von einem gleichverteilten Codebuch mit vier Y-Ebenen und jeweils drei U- und V-Ebenen eine Folge von Codebüchern bei $\lambda = 400$ berechnet. Die Folge zeigt auch in diesem Fall konvergentes Verhalten. Die Codierung der Testsequenzen „Car Phone“ (**Abbildung 4.8**) und „Salesman“ (**Abbildung 4.9**) erfolgt für mehrere Lagrange-Parameter um das Rate-Distortion-Verhalten der Codebuch-Folge zu verdeutlichen.

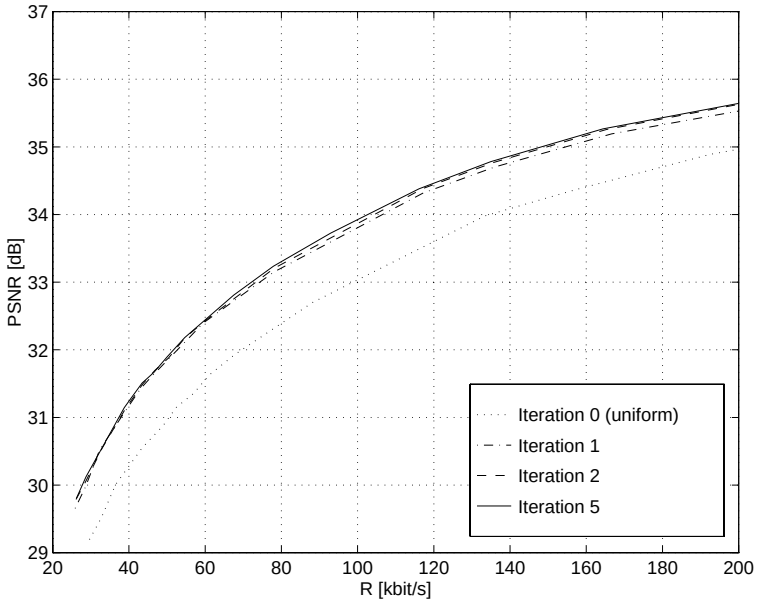


Abbildung 4.8: *Konvergenz des iterativen Algorithmus für den Codebuch-Entwurf mit $\lambda = 400$, vier Y-Ebenen und CIF-Trainingssequenzen. Die Testsequenz „Car Phone“ (CIF, 7.5 fps, 10 s) wurde mit verschiedenen λ-Werten codiert.*

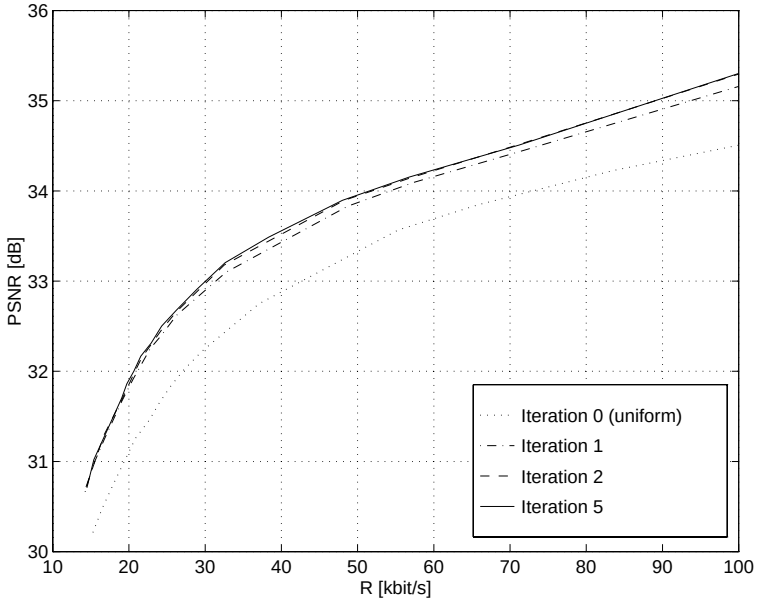


Abbildung 4.9: *Konvergenz des iterativen Algorithmus für den Codebuch-Entwurf mit $\lambda = 400$, vier Y-Ebenen und CIF-Trainingssequenzen. Die Testsequenz „Salesman“ (CIF, 7.5 fps, 10 s) wurde mit verschiedenen λ-Werten codiert.*

Für die CIF-Testsequenzen „Car Phone“ und „Salesman“ hat sich das Rate-Distortion-Verhalten des Quasi-Fixpunkt-Codebuchs gegenüber dem des initialen Codebuchs um ca. 1 dB bzw. 0.8 dB verbessert.

4.4 Optimalität bei verschiedenen Lagrange-Parametern

Der CLGA generiert in Abhängigkeit vom Lagrange-Parameter λ verschiedene Fixpunkt-Codebücher, die nur bei diesem λ-Wert (lokal) optimal sind. Um den Einfluß der verschiedenen Fixpunkt-Codebücher auf das Rate-Distortion-Verhalten der Testsequenzen zu untersuchen, sind mit den QCIF-Trainingsequenzen bei 7.5 fps drei Quasi-Fixpunkt-Codebücher bei $\lambda = 0, 150, 400$ entworfen. In allen drei Fällen ist die Konvergenz der Codebuch-Folge zu beobachten.

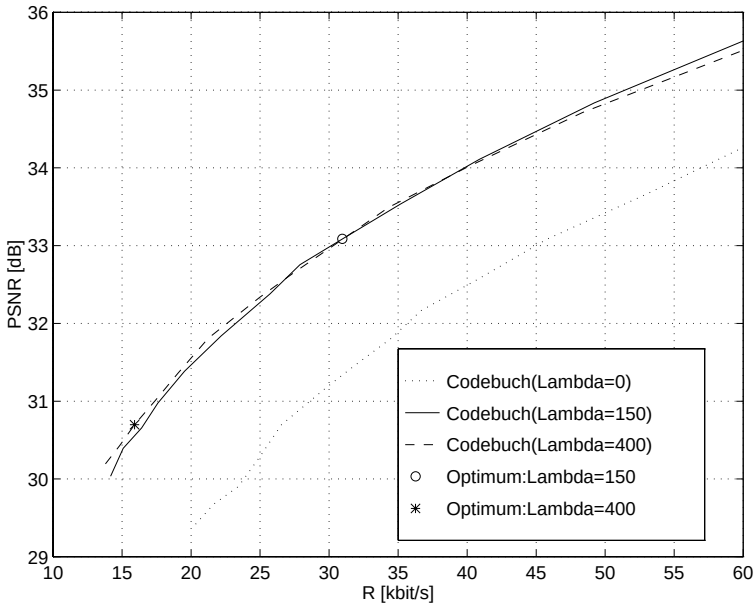


Abbildung 4.10: *Resultate für den Entwurf mit verschiedenen Lagrange-Parametern. Die Testsequenz „Car Phone“ (QCIF, 7.5 fps, 10 s) ist mit dem Pruning-Algorithmus codiert.*

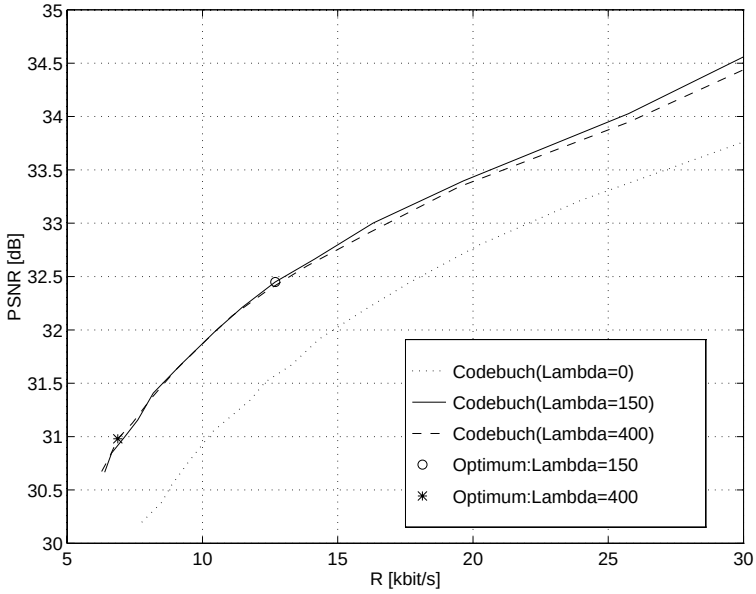


Abbildung 4.11: *Resultate für den Entwurf mit verschiedenen Lagrange-Parametern. Die Testsequenz „Salesman“ (QCIF, 7.5 fps, 10 s) ist mit dem Pruning-Algorithmus codiert.*

Im Fall $\lambda = 0$ wird die Raten-Nebenbedingung ignoriert. Da der verwendete Algorithmus keinen Verbesserungsschritt für die Abbildung β berücksichtigt, ist nur eine Iteration

notwendig um das Fixpunkt-Codebuch für $\lambda = 0$ zu bestimmen. In diesem Fall vereinfacht sich der CLGA zum GLA.

Abbildung 4.10 und 4.11 fassen die Resultate für den Codebuch-Entwurf mit verschiedenen Lagrange-Parametern anhand der QCIF-Testsequenzen „Car Phone“ und „Salesman“ zusammen. Die optimalen Punkte der Quasi-Fixpunkt-Codebücher sind gekennzeichnet. Die in diesem Sinn nicht optimalen Codebücher weisen jeweils ein schlechteres Rate-Distortion-Verhalten auf. Das Fixpunkt-Codebuch ohne Raten-Nebenbedingung ist für niedrige Bitraten nicht geeignet. Dahingegen unterscheiden sich die Quasi-Fixpunkt-Codebücher für $\lambda = 150$ und $\lambda = 400$ in ihrem Rate-Distortion-Verhalten kaum. Somit sind die mit dem iterativen Algorithmus entworfenen Quasi-Fixpunkt-Codebücher in der Umgebung ihres Optimums quasi-optimal. **Tabellen 4.1 – 4.4** geben eine Übersicht über die Dimensionierung des initialen Codebuchs und der Quasi-Fixpunkt-Codebücher.

Level	Shape	Spatial Displacement	Intensity Displacement
0	16	1089	0
1	16	1369	77
2	16	961	97
3	0	1369	93

Tabelle 4.1: *Dimensionierung der partiellen initialen Y-Codebücher.*

Level	Shape	Spatial Displacement	Intensity Displacement
0	16	128	0
1	16	1155	57
2	16	961	97
3	0	1369	93

Tabelle 4.2: *Dimensionierung der partiellen Y-Quasi-Fixpunkt-Codebücher für* $\lambda = 0$.

Level	Shape	Spatial Displacement	Intensity Displacement
0	16	703	0
1	16	1354	77
2	16	961	87
3	0	1281	61

Tabelle 4.3: *Dimensionierung der partiellen Y-Quasi-Fixpunkt-Codebücher für* $\lambda = 150$.

Level	Shape	Spatial Displacement	Intensity Displacement
0	12	667	0
1	16	1360	65
2	14	961	61
3	0	496	61

Tabelle 4.4: *Dimensionierung der partiellen Y-Quasi-Fixpunkt-Codebücher für* $\lambda = 400$.

Alle Ergebnisse in diesem Kapitel sind durch die Codierung der Test- und Trainingssequenzen mit dem Pruning-Algorithmus erzielt.

Kapitel 5

Untersuchung des hierarchischen MIMs

5.1 Einführung

Kapitel 3 behandelt Algorithmen zur Optimierung von Baumstrukturen bei gegebenem Codebuch. **Kapitel 4** erläutert den iterativen Algorithmus zur Bestimmung des optimalen Codebuchs ohne näher auf die Algorithmen zur Bitverteilung einzugehen. In diesem Kapitel untersuchen wir nun beide Aspekte gemeinsam und versuchen damit Eigenschaften des hierarchischen Bewegungs- und Intensitätsmodells darzulegen.

Für die Optimierung der Baumstrukturen nehmen wir die Unabhängigkeit der Modellparameter zwischen Blöcken verschiedener Größen an (siehe **Gleichung 3.24**) und realisieren somit unabhängige Quantisierer für jede Hierarchiestufe.

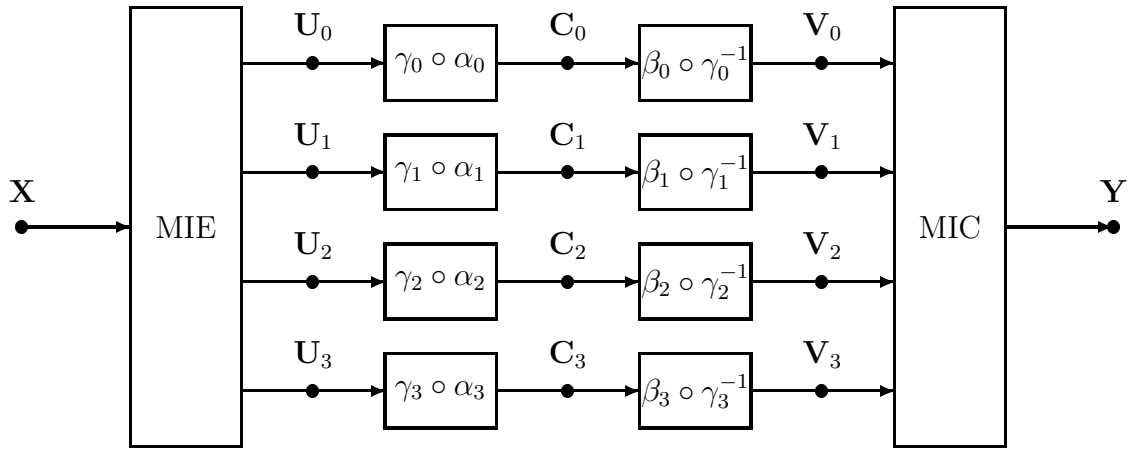


Abbildung 5.1: *Hierarchisches MIM*

Abbildung 5.1 zeigt ein hierarchisches Bewegungs- und Intensitätsmodell mit vier Hierarchiestufen und ihren individuellen Quantisierern $\beta_l \circ \alpha_l$. Mit diesem Modell kann somit jeder Ebene eine partielle Rate zugeordnet werden; dabei ergibt sich die Gesamtrate aus

der Summe der partiellen Raten.

$$R(\mathbf{C}) = \sum_{l=0}^{d-1} R(\mathbf{C}_l) \tag{5.1}$$

Wir untersuchen für die QCIF-Testsequenzen „Car Phone“ und „Salesman“ bei 7.5 fps die Aufteilung der partiellen Raten auf die einzelnen Hierarchiestufen aller Farbkomponenten. Für die Darstellung tragen wir die kumulierten partiellen Raten auf, d.h. wir addieren für jede Rate die partiellen Raten sukzessiv, bis wir die Gesamtrate als Winkelhalbierende kennzeichnen können. Das verwendete Codebuch ist mit den QCIF-Trainingssequenzen und Pruning-Algorithmus bei $\lambda = 150$ optimiert.

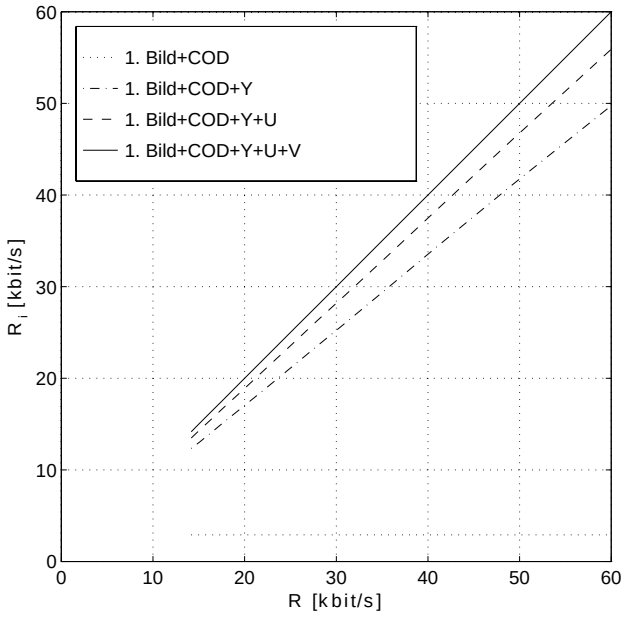


Abbildung 5.2: *Aufteilung der partiellen Raten auf die Farbkomponenten Y, U und V für die Testsequenz „Car Phone“ (7.5 fps, 10 s, QCIF).*

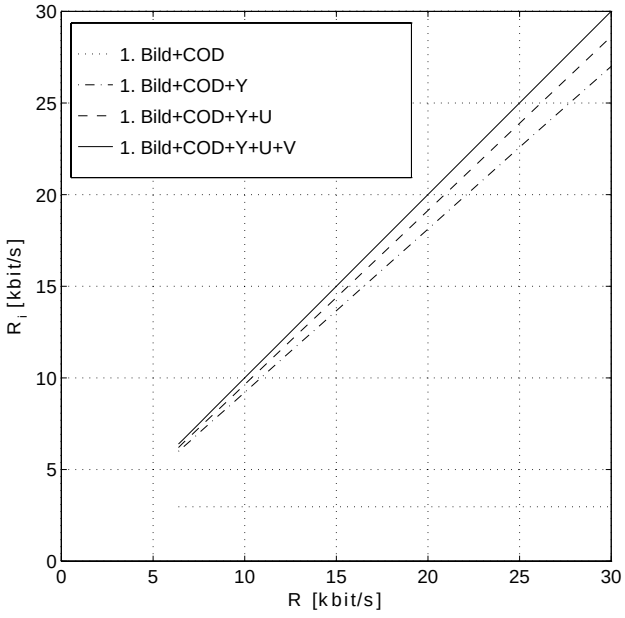


Abbildung 5.3: *Aufteilung der partiellen Raten auf die Farbkomponenten Y, U und V für die Testsequenz „Salesman“ (QCIF, 7.5 fps, 10 s).*

Abbildungen 5.2 und 5.3 veranschaulichen die Aufteilung der partiellen Raten auf die Farbkomponenten Y, U und V anhand der mit dem Pruning-Algorithmus codierten

QCIF-Testsequenzen. Es existieren zwei ratenunabhängige Anteile. Zum einen wird für alle codierten Sequenzen das gleiche erste Bild verwendet und zum anderen ist die Anzahl der Makroblöcke für alle Bilder konstant. Die Rate der Y-Komponente dominiert deutlich über der der U- und V-Komponenten, die sich ihrerseits untereinander ausgewogen darstellen.

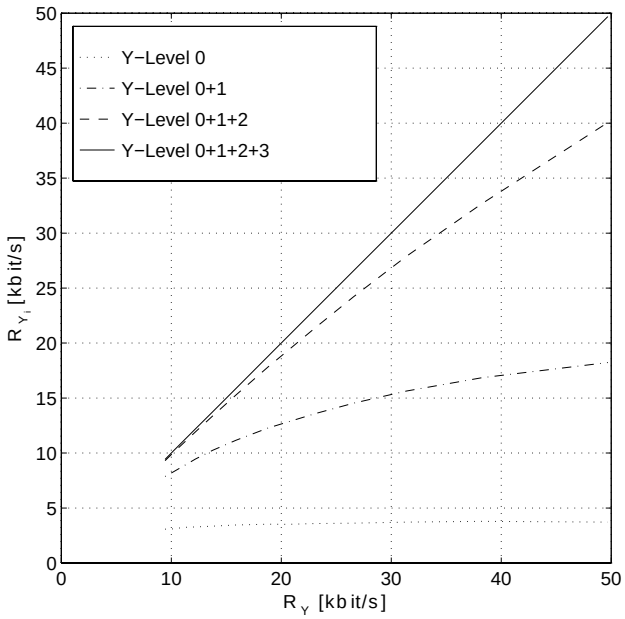


Abbildung 5.4: *Aufteilung der partiellen Raten der Y-Farbkomponente auf die Hierarchiestufen für die Testsequenz „Car Phone“ (QCIF, 7.5 fps, 10 s).*

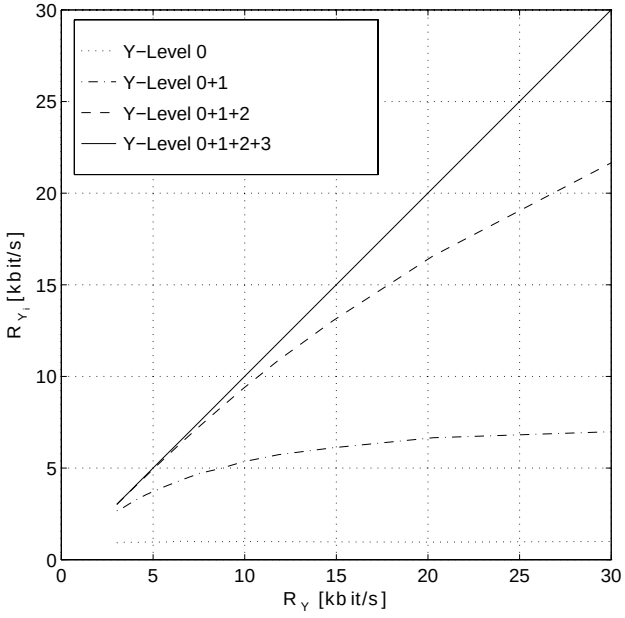


Abbildung 5.5: *Aufteilung der partiellen Raten der Y-Farbkomponente auf die Hierarchiestufen für die Testsequenz „Salesman“ (QCIF, 7.5 fps, 10 s).*

Die Aufteilung der partiellen Raten der Y-Komponente auf die einzelnen Hierarchiestufen ist für die QCIF-Testsequenzen in **Abbildung 5.4 und 5.5** zu finden. Für die Y-Komponente sind vier Hierarchiestufen zugelassen.

1. Bei niedrigen Datenraten wird die tiefste Ebene kaum für die Datenübertragung verwendet. Es ergeben sich „flache“ Baumstrukturen, die der Growing-Algorithmus quasi-optimal bestimmen kann.

2. Für zunehmende Datenraten wird die Bewegungs- und Intensitätskompensation für kleine Blöcke relevant. Andererseits beobachtet man ein Sättigungsverhalten der partiellen Raten für Ebene Null und Eins.

Die maximale Rate des partiellen Kanals der Ebene Null ist relativ klein; besonders für die Testsequenz „Salesman“, bei der das Bewegungsmodell für große Segmente (Intensitätsdifferenz identisch Null) kaum für die Codierung genutzt werden kann.

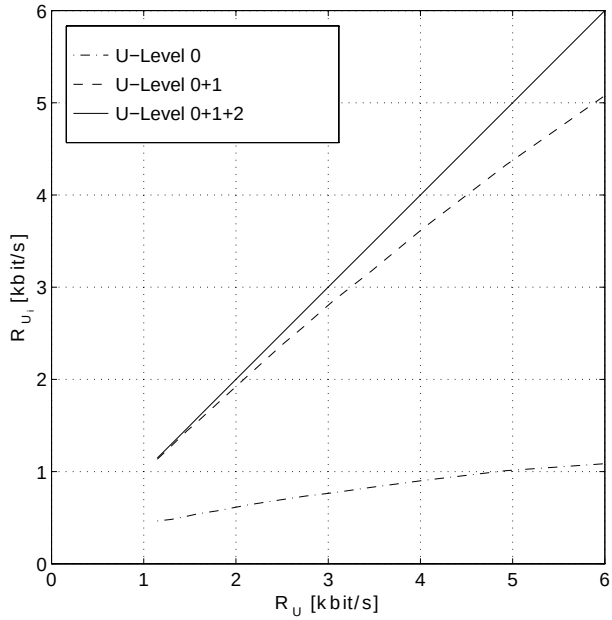


Abbildung 5.6: *Aufteilung der partiellen Raten der U-Farbkomponente auf die Hierarchiestufen für die Testsequenz „Car Phone“ (QCIF, 7.5 fps, 10 s).*

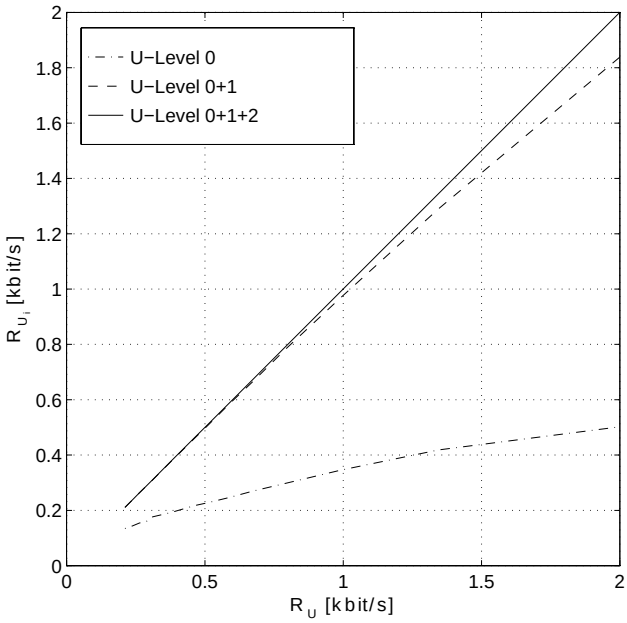


Abbildung 5.7: *Aufteilung der partiellen Raten der U-Farbkomponente auf die Hierarchiestufen für die Testsequenz „Salesman“ (QCIF, 7.5 fps, 10 s).*

Die Aufteilung der partiellen Raten der U-Komponente auf die einzelnen Hierarchiestufen ist für die QCIF-Testsequenzen in **Abbildung 5.6 und 5.7** zu finden. Für die U-Komponente sind drei Hierarchiestufen zugelassen.

1. Für den partiellen Kanal der Ebene Null wird nur die Information der Segmentform übertragen. Die örtliche Verschiebung wird von der nullten Ebene der Y-Komponente vererbt, wobei die Intensitätsdifferenz zu Null angenommen wird.

2. Der Ratenanteil der ersten Ebene ist für das dargestellte Ratenintervall dominant. Hier werden Informationen zur Segmentform, örtlichen Verschiebung und Intensitätsdifferenz übertragen.

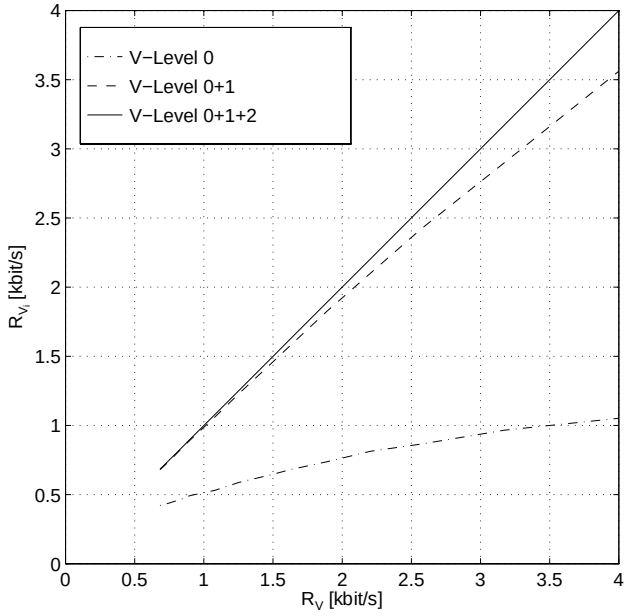


Abbildung 5.8: *Aufteilung der partiellen Raten der V-Farbkomponente auf die Hierarchiestufen für die Testsequenz „Car Phone“ (QCIF, 7.5 fps, 10 s).*

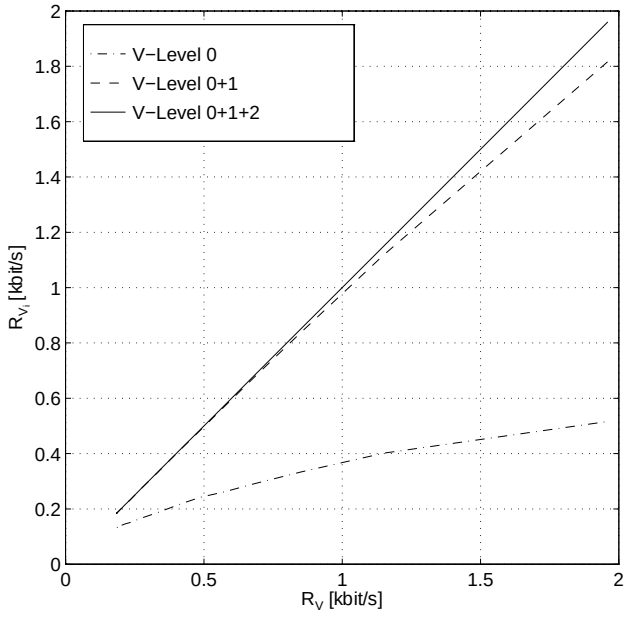


Abbildung 5.9: *Aufteilung der partiellen Raten der V-Farbkomponente auf die Hierarchiestufen für die Testsequenz „Salesman“ (QCIF, 7.5 fps, 10 s).*

Die Aufteilung der partiellen Raten der V-Komponente auf die einzelnen Hierarchiestufen ist für die QCIF-Testsequenzen in **Abbildung 5.8 und 5.9** zu finden. Für die V-Komponente sind drei Hierarchiestufen zugelassen. Es gelten sinngemäß die gleichen Aussagen wie für die U-Komponente.

5.2 Suboptimale Baumstrukturen

In diesem Abschnitt untersuchen wir den Einfluß der suboptimalen Bitverteilung durch den Growing-Algorithmus auf den Codebuch-Entwurf mit dem iterativen Algorithmus. Dazu entwerfen wir ein Codebuch mit dem Growing-Algorithmus und ein Referenz-Codebuch mit dem Pruning-Algorithmus. Für beide Entwürfe wählen wir identische Startbedingungen:

1. konstanter Lagrange-Parameter $\lambda_0 = 150$,
2. konstante Trainingsmenge mit achzehn Trainingssequenzen,
3. initiales Codebuch (β^0, γ^0).

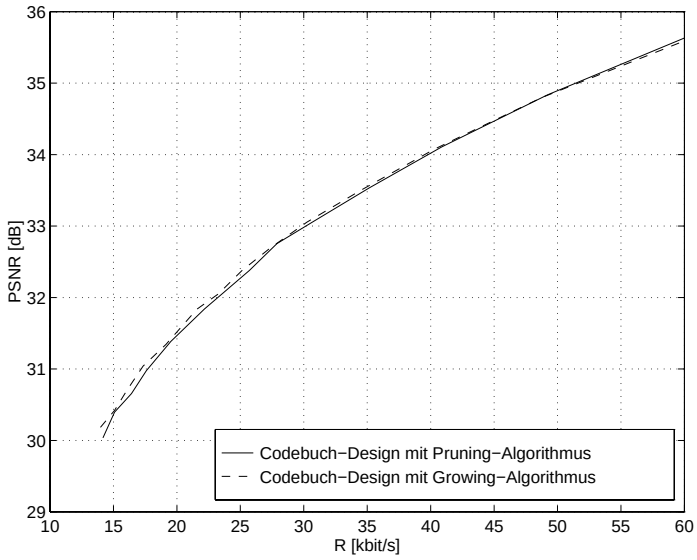


Abbildung 5.10: *Abhängigkeit des Codebuch-Entwurfs mit QCIF-Trainingssequenzen und $\lambda = 150$ von der Bitverteilungsstrategie. Die Testsequenz „Car Phone“ (QCIF, 7.5 fps, 10 s) ist mit dem Pruning-Algorithmus codiert.*

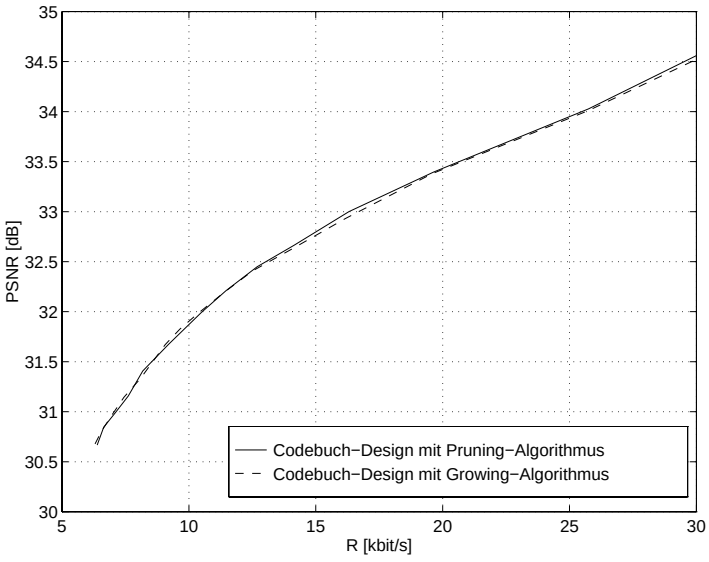


Abbildung 5.11: *Abhängigkeit des Codebuch-Entwurfs mit QCIF-Trainingssequenzen und $\lambda = 150$ von der Bitverteilungsstrategie. Die Testsequenz „Salesman“ (QCIF, 7.5 fps, 10 s) ist mit dem Pruning-Algorithmus codiert.*

Für den Codebuch-Entwurf erhalten wir Folgen von Lagrange-Kosten

$$\begin{aligned} J_P^\kappa &= J(\alpha_P^\kappa, \beta_P^\kappa, \gamma_P^\kappa, \lambda_0, \mathbf{U}) \\ J_G^\kappa &= J(\alpha_G^\kappa, \beta_G^\kappa, \gamma_G^\kappa, \lambda_0, \mathbf{U}), \end{aligned}$$

die in **Abbildung 5.14** dargestellt sind.

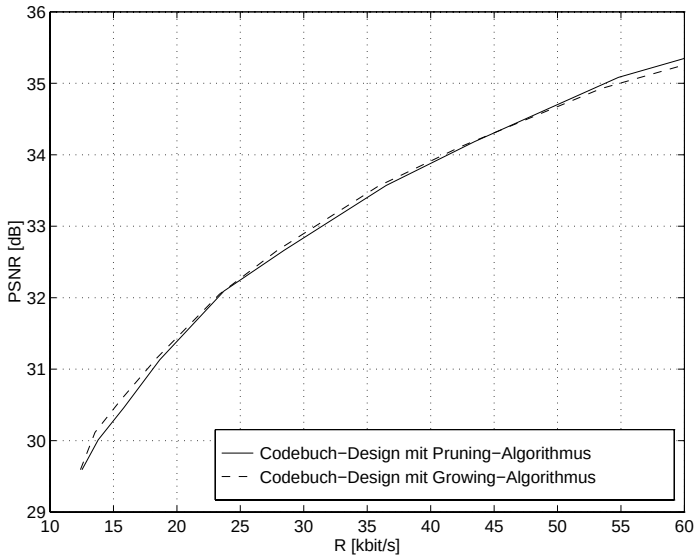


Abbildung 5.12:*Abhängigkeit des Codebuch-Entwurfs mit QCIF-Trainingssequenzen und $\lambda = 150$ von der Bitverteilungsstrategie. Die Testsequenz „Car Phone“ (QCIF, 7.5 fps, 10 s) ist mit dem Growing-Algorithmus codiert.*

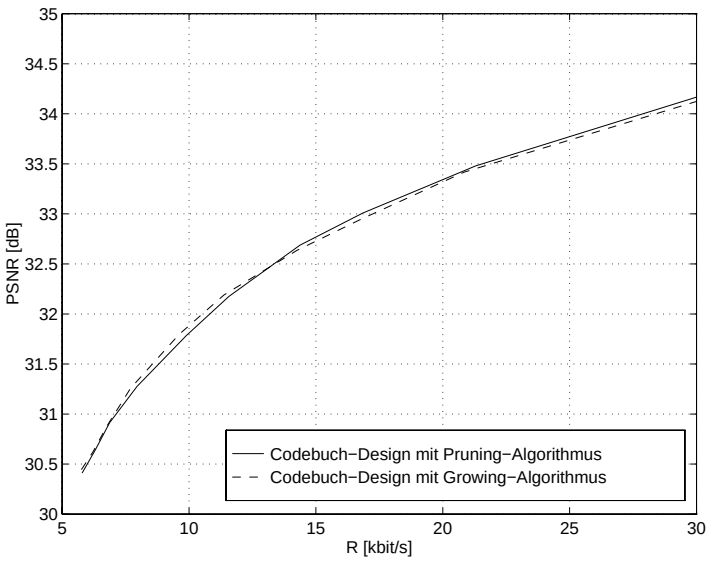


Abbildung 5.13:*Abhängigkeit des Codebuch-Entwurfs mit QCIF-Trainingssequenzen und $\lambda = 150$ von der Bitverteilungsstrategie. Die Testsequenz „Salesman“ (QCIF, 7.5 fps, 10 s) ist mit dem Growing-Algorithmus codiert.*

Für den Vergleich zwischen dem mit dem Pruning-Algorithmus entworfenen Codebuch (β_P, γ_P) und dem mit dem Growing-Algorihmus entworfenen Codebuch (β_G, γ_G) codieren wir die Testsequenzen „Car Phone“ und „Salesman“ mit dem Pruning- bzw. Growing-Algorithmus. Die Ergebnisse zeigen **Abbildungen 5.10 und 5.11** bzw. **Abbildungen 5.12 und 5.13**. Es ist zu erkennen, daß die suboptimale Bitverteilungsstrategie im Rahmen der Simulationsgenauigkeit keinen Einfluß auf den Codebuch-Entwurf aufweist. Dabei ist zu beachten, daß die Codebücher für $\lambda_0 = 150$ optimiert sind. In diesem Bereich weisen Pruning- und Growing-Algorithmus annähernd gleiches Rate-Distortion-Verhalten auf (Siehe **Abbildungen 3.10 und 3.11**).

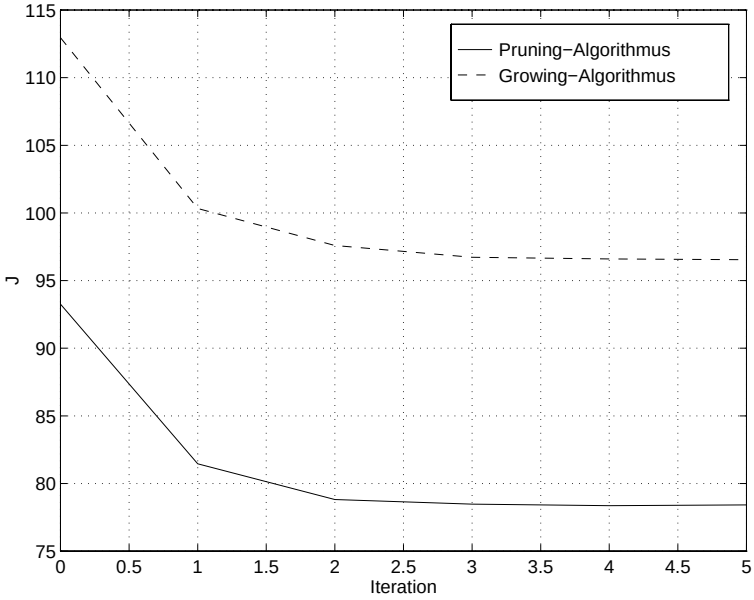


Abbildung 5.14: *Konvergenz des iterativen Algorithmus für den Codebuch-Entwurf mit Pruning- bzw. Growing-Algorithmus,* $\lambda = 150$ *und QCIF-Trainingssequenzen.*

5.3 Vereinfachte Baumstrukturen

In diesem Abschnitt untersuchen wir das Rate-Distortion-Verhalten eines vereinfachten Bewegungs- und Intensitätsmodells. Dazu reduzieren wir die Anzahl der zulässigen Segmentformen von sechzehn auf zwei und verwenden nur noch die Segmente 0 und 15 (Siehe **Abbildung 2.2**). Dies ist gleichbedeutend mit einer Reduktion der Komplexität der codierten Baumstrukturen.

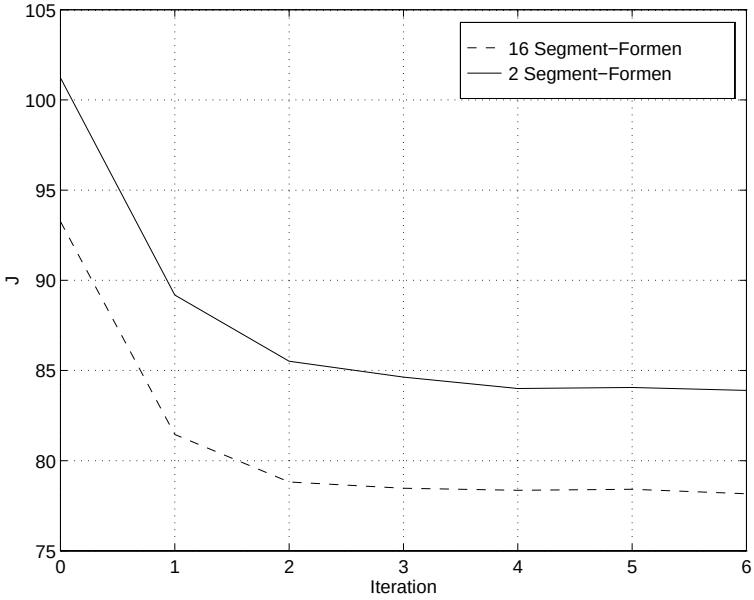


Abbildung 5.15: *Konvergenz des iterativen Algorithmus für den Codebuch-Entwurf mit* $\lambda = 150$*, den QCIF-Trainingssequenzen und dem Pruning-Algorithmus. Die Codebücher unterscheiden sich in der Anzahl der maximal zugelassenen Segmentformen.*

Beide Codebücher sind mit

1. konstantem Lagrange-Parameter $\lambda_0 = 150$,
2. konstanter Trainingsmenge mit achzehn Trainingssequenzen und

3. Pruning-Algorithmus

entworfen. Die Folge der Lagrange-Kosten J_{16}^{κ} und J_{2}^{κ} sind in **Abbildung 5.15** dargestellt.

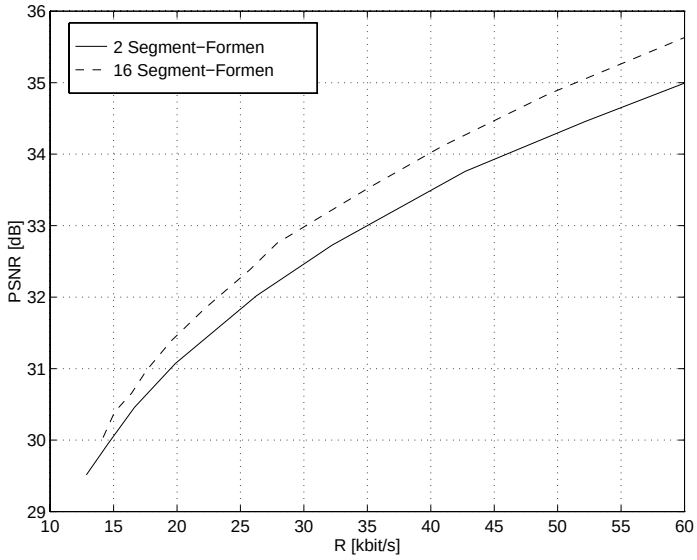


Abbildung 5.16: *Vergleich der Codebücher mit dem Pruning-Algorithmus anhand der Testsequenz „Car Phone“ (QCIF, 7.5 fps, 10 s).*

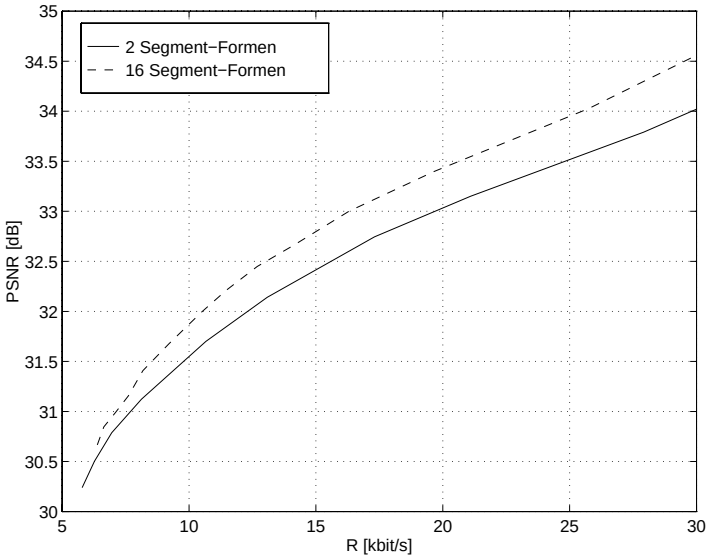


Abbildung 5.17: *Vergleich der Codebücher mit dem Pruning-Algorithmus anhand der Testsequenz „Salesman“ (QCIF, 7.5 fps, 10 s).*

Abbildungen 5.16 und 5.17 zeigen die Codierergebnisse mit dem Pruning-Algorithmus für die zu vergleichenden Codebücher $(\beta_{16}, \gamma_{16})$ und (β_2, γ_2) anhand der Testsequenzen „Car Phone“ und „Salesman“. Bei $40\frac{kbit}{s}$ bzw. $20\frac{kbit}{s}$ ist ein Gewinn von ca. 0.5 dB für das MIM mit sechzehn Segmentformen gegenüber dem MIM mit zwei Segmentformen festzustellen. Für sehr niedrige Datenraten konvergiert der Gewinn gegen Null.

Beim Codebuchentwurf für sehr niedrige Datenraten reduziert sich die Codebuchgröße. Da die zwei Segmentformen eine Teilmenge der sechzehn Segmentformen darstellen, ist das Rate-Distortion-Verhalten der beiden Codebücher für $R \to 0$ identisch.

Da wir aber die Codebücher für $\lambda = 150$ entworfen haben, reduziert sich für sehr niedrige Datenraten die *effektive* Codebuchgröße, d.h. Segmente mit langen Codeworten werden bei der Codierung für sehr niedrige Datenraten mit abnehmender Wahrscheinlichkeit

verwendet. Für $R \to 0$ ist somit das Rate-Distortion-Verhalten der beiden Codebücher vergleichbar.

5.4 Variation der Baumtiefe

Motiviert durch das hierarchische MIM nach **Abbildung 5.1** und die Ergebnisse über die Aufteilung der partiellen Raten auf die Hierarchiestufen (**Abbildungen 5.4 bis 5.9**) untersuchen wir das Verhalten des MIM für variierende minimal zulässige Blockgrößen bei konstanter Größe des Y-Makroblocks von 32×32 bzw. der U- und V-Makroblöcke von 16×16.

Dazu entwerfen wir Codebücher mit dem iterativen Algorithmus für 3, 4 und 5 Y-Ebenen, d.h. die kleinsten zulässigen Blockgrößen für die Y-Komponente betragen jeweils 8×8, 4×4 und 2×2. Die U- und V-Komponenten besitzen jeweils eine Ebene weniger als die dazu korrespondierende Y-Komponente.

Alle Codebücher werden mit den folgenden Bedingungen bestimmt:

1. konstanter Lagrange-Parameter $\lambda_0 = 400$,
2. konstante Trainingsmenge mit achzehn CIF-Trainingssequenzen,
3. Pruning-Algorithmus,
4. sechzehn Segmentformen sind zulässig.

Die Folgen der Lagrange-Kosten J_3^κ, J_4^κ undJ_5^κ sind in **Abbildung 5.18** dargestellt.

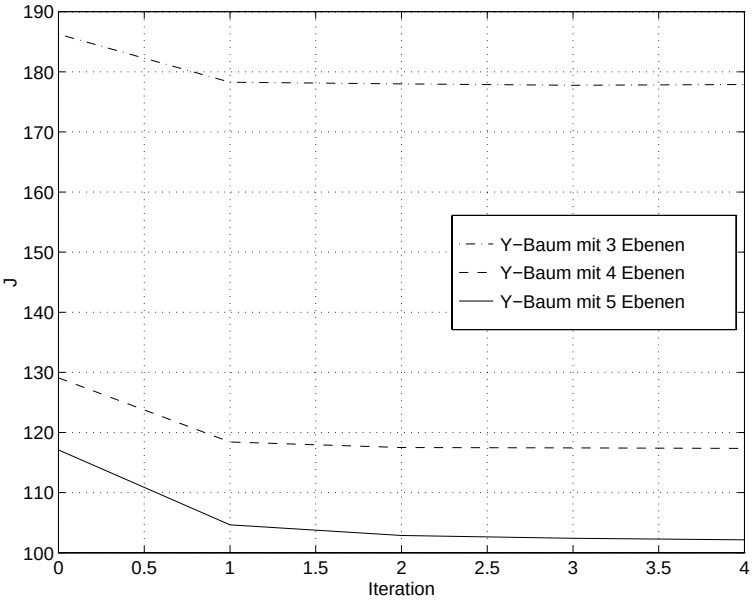


Abbildung 5.18: *Konvergenz des iterativen Algorithmus für den Codebuch-Entwurf mit $\lambda = 400$, Pruning-Algorithmus und CIF-Trainingssequenzen. Die Codebücher unterscheiden sich bei konstanter Makroblockgröße um die kleinste zugelassene Blockgröße.*

Abbildungen 5.19 und 5.20 zeigen die Codierergebnisse mit dem Pruning-Algorithmus für die zu vergleichenden Codebücher (β_3, γ_3), (β_4, γ_4) und (β_5, γ_5) anhand der Testsequenzen „Car Phone“ und „Salesman“ in CIF-Auflösung.

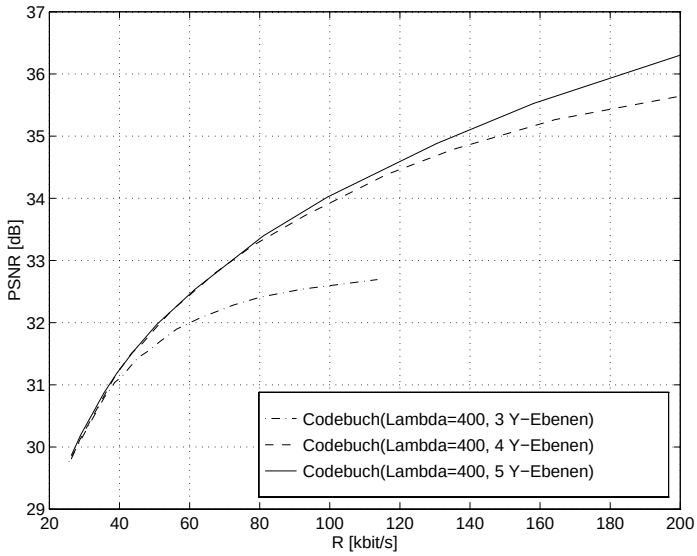


Abbildung 5.19: *Vergleich zwischen Baumstrukturen variierender maximaler Baumtiefe anhand der mit dem Pruning-Algorithmus codierten Testsequenz „Car Phone“ (CIF, 7.5 fps, 10 s).*

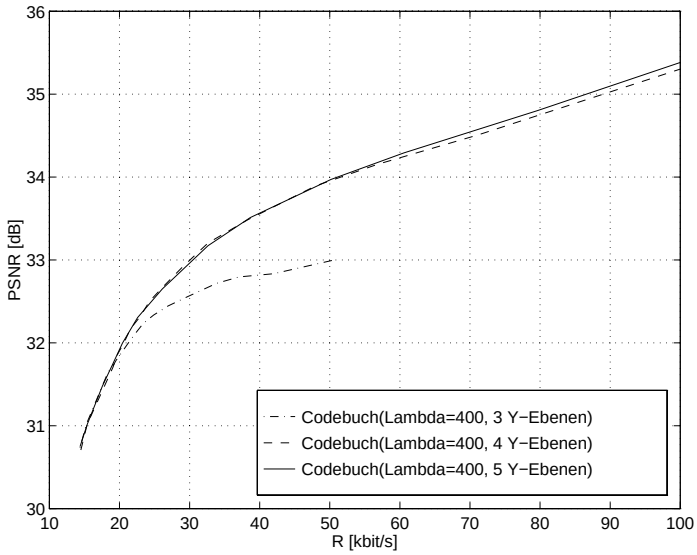


Abbildung 5.20: *Vergleich zwischen Baumstrukturen variierender maximaler Baumtiefe anhand der mit dem Pruning-Algorithmus codierten Testsequenz „Salesman“ (CIF, 7.5 fps, 10 s).*

Es ist festzustellen, daß

1. für zunehmende maximale Baumtiefe Codiergewinne für höhere Datenraten möglich werden. Mit dem hierarchischen MIM nach **Abbildung 5.1** ist dies so zu erklären, daß für hohe Datenraten die Kanäle der tieferen Ebenen stärker genutzt werden können, wenn die darüber liegenden Kanäle bereits ihre maximale Raten übertragen. Die Erhöhung der maximalen Baumtiefe ist aber mit einer Zunahme an Codierkomplexität verbunden.

2. bei sehr niedrigen Datenraten sich die hierarchischen Modelle nicht unterscheiden. Dies ist darauf zurückzuführen, daß die partiellen Raten eventuell vorhandener tieferer Ebenen in diesem Fall gegen Null konvergieren.

5.5 Auflösung der Trainingssequenzen

In der abschließenden Untersuchung entwerfen wir Codebücher mit dem iterativen Algorithmus für verschiedene Auflösungen der Trainingssequenzen. Das CIF-Format mit 352×288 besitzt die vierfache Anzahl an Bildelementen im Vergleich zum QCIF-Format mit 176×144. Bestimmen wir das Codebuch (β_Q, γ_Q) mit QCIF-Trainingssequenzen bei $\lambda_Q = 100$, so haben wir das Codebuch (β_C, γ_C) mit CIF-Trainingssequenzen bei $\lambda_C = 400$ zu generieren um die Rate gegenüber dem quadratischen Fehler korrekt zu gewichten.

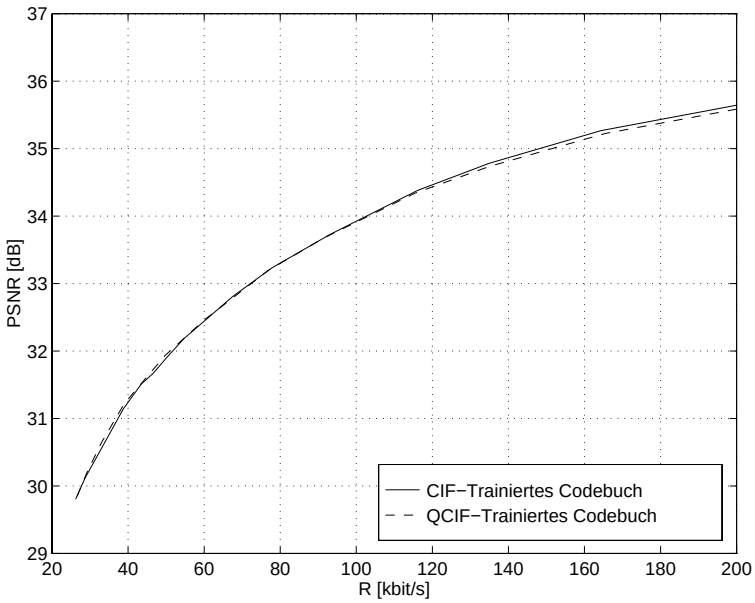


Abbildung 5.21: *Vergleich zwischen CIF- und QCIF-trainierten Codebüchern anhand der Testsequenz „Car Phone“ (CIF, 7.5 fps, 10 s). Die Bitverteilung wird jeweils durch den Pruning-Algorithmus bestimmt.*

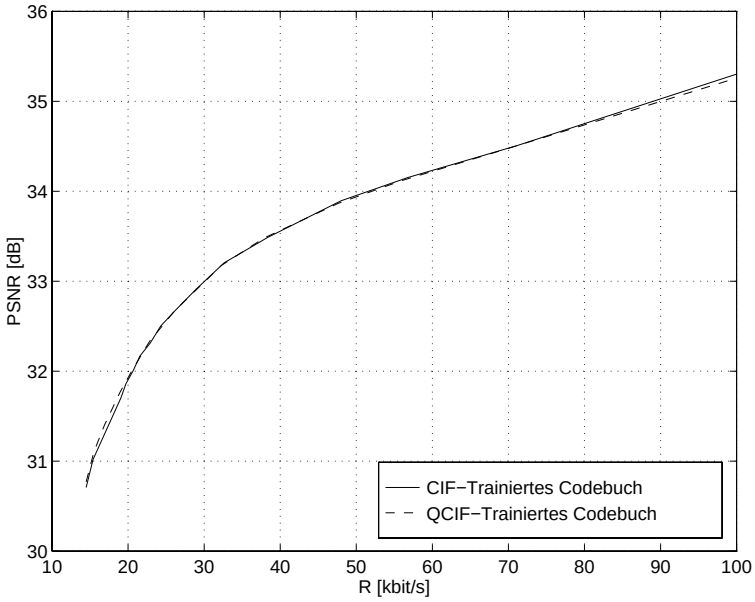


Abbildung 5.22: *Vergleich zwischen CIF- und QCIF-trainierten Codebüchern anhand der Testsequenz „Salesman“ (CIF, 7.5 fps, 10 s). Die Bitverteilung wird jeweils durch den Pruning-Algorithmus bestimmt.*

Außerdem treffen folgende Randbedingungen für beide Codebücher zu:

1. konstante Trainingsmenge weist identischen Inhalt bei verschiedener örtlicher Auflösung auf,

2. Pruning-Algorithmus,

3. konstantes hierarchisches Modell mit sechzehn Segmentformen.

Abbildungen 5.21 und 5.22 zeigen die Codierergebnisse mit dem Pruning-Algorithmus für die zu vergleichenden Codebücher (β_Q, γ_Q) und (β_C, γ_C) anhand der Testsequenzen „Car Phone“ und „Salesman“ in CIF-Auflösung. Es ist festzustellen, daß im Rahmen der Simulationsgenauigkeit beide Codebücher gleiches Rate-Distortion-Verhalten für beide CIF-Testsequenzen aufweisen. Dies ist damit zu begründen, daß ein bewegungs- und intensitätskompensierter Block der Größe 16×16 in einem QCIF-Bild mit den gleichen Parametern (örtliche Verschiebung und Intensitätsdifferenz) codiert wird wie der der Größe 32×32 in einem CIF-Bild. Somit werden bei der Codierung auf jeder Hierarchiestufe die gleichen Parameter bei einer verschiedenen örtlichen Auflösung extrahiert.

Kapitel 6

Ausblick

Die untersuchten Algorithmen zur Bestimmung von Quadtree-Strukturen bei der Videocodierung mit Blöcken variabler Größe bieten an vielen Stellen Möglichkeiten zur Verbesserung. Dazu gehören sicher die Berücksichtigung des Overlappings bei der Codierung und die Weiterentwicklung eines Growing-Algorithmus, bei dem ein lokaler Pruning-Algorithmus die suboptimale Segmentform bestimmt. Viel interessanter hingegen scheint aber die Analyse des hierarchischen Bewegungs- und Intensitätsmodells und der Entwurf des Modell-Quantisierers mit Raten-Nebenbedingung.

6.1 Untersuchung der Skalierbarkeit

Die absolute Auflösung der Blöcke auf den Ebenen ist für die Codierung/Decodierung irrelevant. Die relative Auflösung zwischen den Hierarchiestufen und die maximal zulässige Baumtiefe ist dagegen immanent. In **Abschnitt 5.5** haben wir gezeigt, daß die gleiche Modellstatistik unabhängig von der Auflösung der Trainingssequenzen gewonnen werden kann. Die so entstandenen Codebücher sind sich also ähnlich. Somit ist es auch möglich, eine Codesequenz, die aus einer QCIF-Videoseqenz entstanden ist, im CIF-Format zu decodieren. Dabei ist für vorgegebene absolute Makroblock-Größe nur der absolute Wert der örtlichen Verschiebungen zu skalieren.

6.2 Modifikation des MIM

Das Bewegungs- und Intensitätsmodell nach **Gleichung 2.1** basiert auf der örtlichen Verschiebung der Blöcke und Addition der Intensitätsdifferenz. Durch die Suche nach einem verbesserten Quellenmodell motiviert, schlagen wir als Variation eine *Skalierung* der Intensitätswerte nach **Gleichung 6.1** vor.

$$\mathbf{y}[m,n,k] = \mathbf{a}[m,n,k] \cdot \mathbf{y}[m+\mathbf{\Delta m}[m,n,k], n+\mathbf{\Delta n}[m,n,k], k-1] \qquad (m,n) \in S[k] \quad (6.1)$$

Um jedoch die beiden Modelle nach **Gleichung 2.1 und 6.1** tatsächlich vergleichen zu können, sollte man örtliche Verschiebung und Intensitätsdifferenz bzw. Intensitäts-

skalierung zu einem drei-dimensionalen Ereignis $(\mathbf{\Delta m}, \mathbf{\Delta n}, \mathbf{q})$ bzw. $(\mathbf{\Delta m}, \mathbf{\Delta n}, \mathbf{a})$ zusammenfassen und gemeinsam codieren. Die mögliche unterschiedliche Suboptimalität der Produkt-Codes könnte die Resultate verfälschen.

Trotzdem sei für das Modell nach **Gleichung 6.1** eine Zerlegung vorgeschlagen. Wir minimieren über alle möglichen örtlichen Verschiebungen.

$$\min_{(\mathbf{\Delta m}, \mathbf{\Delta n})} \sum_{(m,n)\in S} \Big|\mathbf{x}[m,n,k] - \hat{\mathbf{a}}[\mathbf{\Delta m}, \mathbf{\Delta n}]\mathbf{y}[m+\mathbf{\Delta m}, n+\mathbf{\Delta n}, k-1]\Big|^2 + \lambda\Big\{\mathbf{l}(S) + \mathbf{l}(\mathbf{\Delta m}, \mathbf{\Delta n}) + \mathbf{l}(\hat{\mathbf{a}}[\mathbf{\Delta m}, \mathbf{\Delta n}])\Big\}$$

Unabhängig von der Raten-Nebenbedingung bestimmen wir einen Schätzwert für den Skalierungfaktor in Abhängigkeit von der örtlichen Verschiebung.

$$\hat{\mathbf{a}}[\mathbf{\Delta m}, \mathbf{\Delta n}] = Q\left[\frac{\sum_{(m,n)\in S} \mathbf{x}[m,n,k]\mathbf{y}[m+\mathbf{\Delta m}, n+\mathbf{\Delta n}, k-1]}{\sum_{(m,n)\in S} \mathbf{y}^2[m+\mathbf{\Delta m}, n+\mathbf{\Delta n}, k-1]}\right]$$

Eine weitere Variation erhöht die Ordnung des Markov-Modells. Dazu lassen wir eine Bewegungs- und Intensitätskompensation nicht nur vom Bild zur diskreten Zeit $k-1$, sondern von allen Bilder zu den diskreten Zeiten $k-k_0$ bis $k-1$ zu.

$$\mathbf{y}[m,n,k] = \mathbf{y}[m+\mathbf{\Delta m}, n+\mathbf{\Delta n}, k-\mathbf{\Delta k}] + \mathbf{q} \qquad (m,n) \in S[k]$$

Die quantisierten Parameter sollten als vier-dimensionales Ereignis $(\mathbf{\Delta m}, \mathbf{\Delta n}, \mathbf{\Delta k}, \mathbf{q})$ codiert werden.

6.3 Entwurf des Modell-Quantisierers

In **Abschnitt 4.3** wurde der iterative Algorithmus für den Entwurf des Modell-Quantisierers diskutiert. In der vorliegenden Arbeit wird kein Verbesserungsschritt für die Abbildung β vorgenommen. Alle zugelassenen Zentroiden befinden sich auf Gitterpositionen. Wird nun aber die Genauigkeit der Quantisierung um den Faktor F erhöht, so erhöht sich für ein d-Dimensionales Gitter die Anzahl der Gitterpositionen und somit die der Codeworte um den Faktor F^d.

Wir haben ein Codebuch bei $\lambda = 150$ mit QCIF-Trainingssequenzen und Pruning-Algorithmus entworfen, das alle vorher mit Genauigkeit Vier quantisierten Intensitätsdifferenzen jetzt mit Genauigkeit Zwei quantisiert. Dadurch wird die Anzahl der Codeworte um den Faktor Zwei erhöht, d.h. es wird im Mittel 1 bit extra benötigt, um die Zwischenpositionen zu codieren.

Die Codebücher im Vergleich anhand der Testsequenzen „Car Phone“ und „Salesman“ zeigen die **Abbildungen 6.1 und 6.2**. Es ist festzustellen, daß die erhöhte Rate bei der Intensitätsdifferenz nicht durch einen PSNR-Gewinn, der durch eine Erhöhung der Genauigkeit erreicht wird, kompensiert werden kann. Das Rate-Distortion-Verhalten des Codebuchs mit den Zwei-Gittern ist schlechter als das des Codebuchs mit den Vier-Gittern.

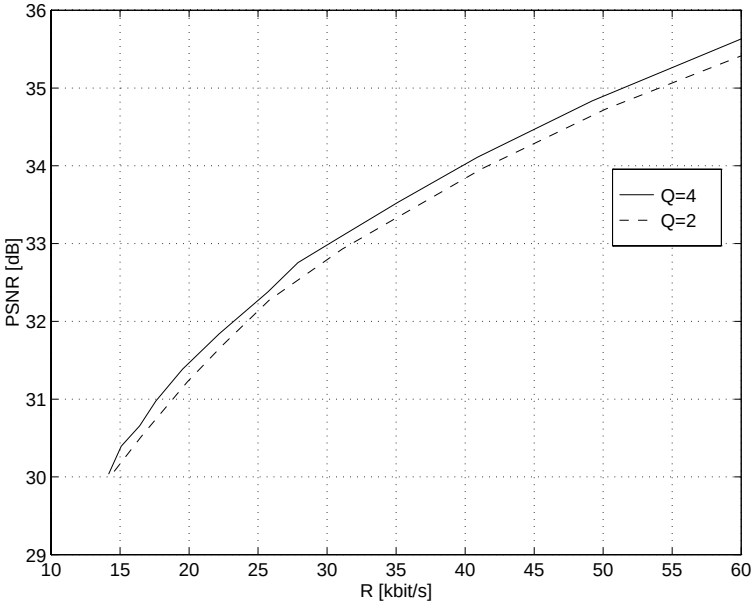


Abbildung 6.1: *Vergleich zwischen Codebüchern mit einem 4- bzw. 2-Gitter für die Intensitätsdifferenz anhand der Testsequenz „Car Phone“ (QCIF, 7.5 fps, 10 s). Die Bitverteilung wird jeweils durch den Pruning-Algorithmus bestimmt.*

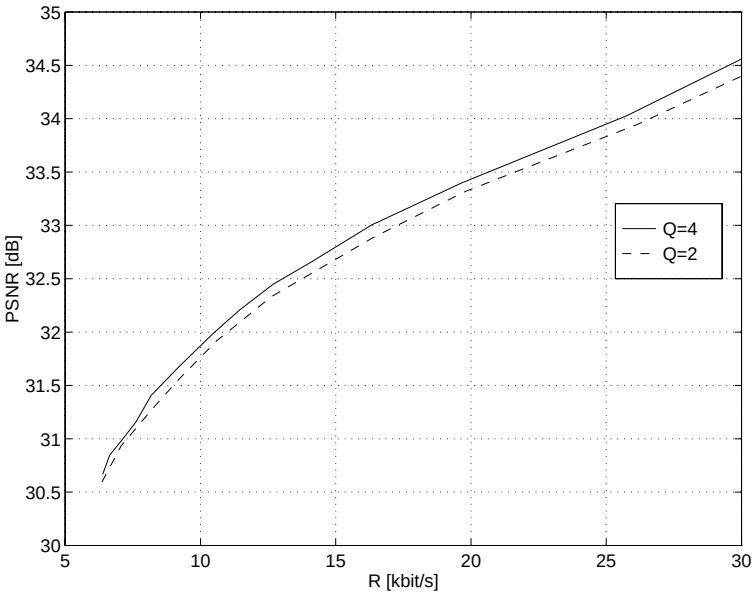


Abbildung 6.2: *Vergleich zwischen Codebüchern mit einem 4- bzw. 2-Gitter für die Intensitätsdifferen anhand der Testsequenz „Salesman“ (QCIF, 7.5 fps, 10 s). Die Bitverteilung wird jeweils durch den Pruning-Algorithmus bestimmt.*

Gish and Pierce haben gezeigt, daß der asymtotische optimale skalare Quantisierer mit Entropie-Nebenbedingung unabhängig von der Verteilung der Quelle äquidistante Zentroiden aufweist. Gersho vermutet für den asymtotischen optimalen Vektorquantisierer mit Entropie-Nebenbedingung die Form eines Gitters [15].

Der iterative Algorithmus aus **Abschnitt 4.3** löst das Problem der optimalen Positionierung der Zentroiden mit Hilfe des Verbesserungsschrittes nach **Gleichung 4.13**. Dadurch sind wir in der Lage, die Voronoi-Regionen des initialen Gitters optimal zu formen und das tatsächliche Bewegungs- und Intensitätsfeld durch das quantisierte Feld zu approximieren.

Anhang A

Kurzbeschreibung des Software-Pakets

Im Rahmen der Untersuchung von Algorithmen zur Bestimmung von Quadtree-Strukturen bei der Videocodierung mit Blöcken variabler Größe sind zahlreiche Module entworfen worden. Coder und Decoder, Module zur Optimiertung des Codebuchs und ein Analyse-Modul werden im folgenden erläutert.

A.1 Variable Block Size Codec

A.1.1 Coder

Der VBS-Coder *(Variable Block Size Coder)* generiert aus der gegebenen Videosequenz eine Codesequenz, deren Datenrate durch Irrelevanz- und Redundanzreduktion verringert ist. Der Coder kann mit folgenden Optionen betrieben werden:

```
VBS coder version 1.0
(C) 1996 University of Erlangen-Nuremberg
    T. Wiegand, M. Flierl

QtEncode [-<help>hiorR<RFF>t<info>CK<ff>nmabsfkDSMdOlzG]
    -help        verbose help
    -h           help
    -i    [name] original sequence file name
                 []
    -o    [name] code file name
                 [bits.itd]
    -r    [name] reconstructed sequence file name
                 []
    -R    [num]  target bit rate [0 bps]
    -RFF  [num]  bits used for first frame [0 bits]
    -t    [name] trace file name
```

```
-info [name] information file name
-C    [name] codebook file name
             [/POOL_HOME/stud/mflierl/Coding/Quadtree/Codec/tab/itd1.0.cbk]
-K    [name] mask file name
             [/POOL_HOME/stud/mflierl/Coding/Quadtree/Codec/tab/mask.tab]
-ff   [name] first frame file name
-n    [num]  input image width       [176]
-m    [num]  input image height      [144]
-a    [num]  start                   [0]
-b    [num]  end                     [99]
-s    [num]  frame step              [1]
-f    [num]  frequency               [30]
-k    [0/1]  khoros format           [0]
-D    [num]  output mode             [2]
               0: write only code file
               1: write reconstruction
               2: display reconstruction
               3: display imposed reconstruction
-S    [num]  code buffer size (byte) [100]
-M    [num]  macroblock size         [16]
-d    [0/1]  differencing            [1]
-O    [0/1]  overlapping             [1]
-l    [num]  lambda                  [100]
-z    [num]  zoom factor             [1]
-G           use growing algorithm
```

Die Option `-i /DATA/sequences/QCIF_CD/mpeg4/coastgua/cg` spezifiziert z.B. die Videosequenz *Coastguard* in QCIF Auflösung, wobei mit `-o cg.itd` der Filename der Codesequenz festgelegt wird. Zusätzlich sind noch die Bildbreite `-n 176`, die Bildhöhe `-m 144`, die Makroblock-Größe `-M 16` (16: QCIF-Bild; 32: CIF-Bild), die Bildrate `-f 30` (30 Bilder pro Sekunde) und das Fileformat `-k 0` (0: LNT-Library-Format; 1: Khoros-Format) des Originals anzugeben. Sollen zum Beispiel die Bilder 0, 4, 8, ... 292, 296 codiert werden, so ist dies mit `-a 0 -s 4 -b 296` zu beschreiben. `-S 100` legt die Größe des Zwischenspeichers für die Codebits auf 100 Byte fest, d.h. die Codesequenz wird in Paketen zu 100 Byte unter dem File `cg.itd` gespeichert.

Mit `-r cg/` wird das Verzeichnis `cg/` bestimmt, in dem die rekonstruierten Bilder gespeichert werden können. Diese sind dann identisch mit den rekonstruierten Bilder, die der Decoder aus der Codesequenz erzeugt, wenn Coder als auch Decoder mit identischen Parametern arbeiten. (In der Version 1.0 enthält die Codesequenz noch keine Steuerinformationen.)

Dieser Codec nutzt das Prinzip der *Motion and Intensity Compensation (MIC)* und benötigt somit ein Startbild. Ohne Zusatzangaben beginnt die Codierung mit einem Graubild. Mit `-ff /DATA/sequences/QCIF_CD/mpeg4/coastgua/cg` und unter Angabe der benötigten Codebits `-RFF 304128` kann dieses extern zugeführt werden; in diesem Beispiel wird der Originalbild mit $176 \cdot 144 \cdot 1.5 \cdot 8$ bits geladen.

Mit der Option `-d 1` wird die Differenz zwischen geschätztem und tatsächlichem Bewegungsvektor codiert, wodurch die örtliche Korrelation der Bewegungsvektoren und somit

auch die Datenrate reduziert wird. `-d 0` setzt die Werte des Bewegungsvektor-Prediktors auf Null.

Im Rahmen dieser Studienarbeit wurde ein *Pruning-Algorithmus* und ein *Growing-Algorithmus* zur Bestimmung von Quadtree-Strukturen implementiert. Ohne explizite Option wird der Pruning-Algorithmus verwendet, wohingegen `-G` den Growing-Algorithmus zum Einsatz kommen läßt.

Der VBS-Coder ist lokal Rate-Distortion optimal, d.h. es wird eine modifizierte Distortion Funktion (Lagrange-Funktion) $J = D + \lambda R$ minimiert. D beschreibt den quadratischen Fehler zwischen Original und Rekonstuktion und R die Länge des Codes in Bit, die notwendig ist, um die Rekonstruktion zu beschreiben. `-l 100` weist dem Optimierungsparameter λ den Wert 100 zu und bleibt konstant für alle Bilder der Videosequenz. Wird hingegen die Ratenregelung mit `-R 24000` aktiviert, so wird der Parameter λ für jedes Bild individuell derart bestimmt, daß die Codesequenz eine Datenrate von 24000 bps aufweist. (In der Version 1.0 ist die Varianz der Abweichung von der Ziel-Datenrate relativ hoch.)

Die Option `-D` legt den Ausgabemodus des Coders fest. Bei jeder Codierung wird die Codesequenz erzeugt. `1` generiert zusätzlich die rekonstruierten Bilder, die im Verzeichnis `cg/` gespeichert werden. `2` visualisiert zusätzlich die rekonstruierten Bilder mit einem Display, die z.B mit `-z 2` um den Faktor Zwei vergrößert dargestellt werden. (als Faktoren dürfen nur natürliche Zahlen auftreten) `3` visualisiert zusätzlich die dem rekonstruierten Bild überlagerte Quadtree-Stuktur der Y-Komponente.

`-t cg.trc` veranlaßt den Coder, ein *Trace File* zu generieren. Dieses File beschreibt die Semantik der Codesequenz und ist hier für die ersten vier codierten Makroblöcke dargestellt:

```
picture_start_code:      at 304128, frame:   4
MB ( 0, 0)  COD: 1       at 304128, wrote "1"
Component 0: (Px,Py)=( 0, 0)
L0: Shape 15     at 304129, wrote "11"
  L1: Shape 15   at 304131, wrote "110"
    L2: Shape  4         at 304134, wrote "0100"
    L2: (x_diff,y_diff)=(  0,  0) (x,y)=(  0,  0)   at 304138, wrote "01"
    L2: qValue   0       at 304140, wrote "0"
      L3: (x_diff,y_diff)=(  0,  0) (x,y)=(  0,  0) at 304141, wrote "10"
      L3: qValue  28     at 304143, wrote "1010001"
    L2: Shape  0         at 304150, wrote "1"
    L2: (x_diff,y_diff)=(  0, -1) (x,y)=(  0, -1)   at 304151, wrote "1100"
    L2: qValue   4       at 304155, wrote "100"
    L2: Shape  0         at 304158, wrote "1"
    L2: (x_diff,y_diff)=( -3,  0) (x,y)=( -3,  0)   at 304159, wrote "0000010"
    L2: qValue   0       at 304166, wrote "0"
    L2: Shape  0         at 304167, wrote "1"
    L2: (x_diff,y_diff)=( -3,  0) (x,y)=( -3,  0)   at 304168, wrote "0000010"
    L2: qValue   0       at 304175, wrote "0"
  L1: Shape  1   at 304176, wrote "0011"
  L1: (x_diff,y_diff)=(  0,  0) (x,y)=(  0,  0)     at 304180, wrote "1"
  L1: qValue   0         at 304181, wrote "0"
```

```
    L2: Shape  0         at 304182, wrote "1"
    L2: (x_diff,y_diff)=(  0, -1) (x,y)=(  0, -1)   at 304183, wrote "1100"
    L2: qValue   0       at 304187, wrote "0"
  L1: Shape  0   at 304188, wrote "10"
  L1: (x_diff,y_diff)=( -3,  0) (x,y)=( -3,  0)     at 304190, wrote "0001111"
  L1: qValue   0         at 304197, wrote "0"
  L1: Shape  0   at 304198, wrote "10"
  L1: (x_diff,y_diff)=( -3,  0) (x,y)=( -3,  0)     at 304200, wrote "0001111"
  L1: qValue   0         at 304207, wrote "0"
Component 1: (Px,Py)=( 0, 0)
L0: Shape  0     at 304208, wrote "0"
Component 2: (Px,Py)=( 0, 0)
L0: Shape  0     at 304209, wrote "0"
MB ( 0, 1)  COD: 1       at 304210, wrote "1"
Component 0: (Px,Py)=( 0, 0)
L0: Shape  0     at 304211, wrote "000"
L0: (x_diff,y_diff)=( -3,  0) (x,y)=( -3,  0)    at 304214, wrote "011100"
Component 1: (Px,Py)=(-1, 0)
L0: Shape  0     at 304220, wrote "0"
Component 2: (Px,Py)=(-1, 0)
L0: Shape  0     at 304221, wrote "0"
MB ( 0, 2)  COD: 0       at 304222, wrote "0"
MB ( 0, 3)  COD: 0       at 304223, wrote "0"
```

Für dieses Beispiel ist das Bild 4 vom geladenen Startbild 0 aus geschätzt. Das Startbild ist mit 304128 bit codiert und der Bit-Zähler beginnt bei 304128 für Bild 4; der Code befindet sich aber nicht in der Codesequenz. Das Bild ist in Makroblöcke aufgeteilt, wobei diese durch `MB(<Zeilenposition>, <Spaltenposition>)` gekennzeichnet werden. Jedem Makroblock ist ein COD-Bit zugeordnet. Im Fall des gesetzten COD-Bits folgt die Quadtree-Beschreibung der Y-, U- und V-Komponenten (0,1,2) in der angegebenen Reihenfolge. Die Quadtree-Beschreibung ist *„depth-first“* strukturiert, d.h. das aktuelle Teilsegment wird weiter unterteilt, bevor das nächste Teilsegment auf dem aktuellen Level beschrieben wird. Ein Teilsegment ist durch seine Form (*Shape*), örtliche Verschiebung (*Spatial Displacement*) und Intensitätsdifferenz (*Intensity Displacement*) festgelegt. `Shape` kennzeichnet die 16 möglichen Segmentformen und `qValue` die Intensitätsdifferenz. Für die örtliche Verschiebung kennzeichnet `(Px,Py)` den Schätzwert des Bewegungsvektors, `(x,y)` den tatsächlichen Wert des Bewegungsvektors und `(x_diff,y_diff)` die Differenz, der schließlich ein Codewort zugeordnet wird. Es ist zu bemerken, daß nicht für jedes Segment zwingend alle drei Parameter codiert werden müssen; auf dem tiefsten Level ist z.B. der Parameter `Shape` nicht notwendig, da er in jedem Fall Null ist.

`-info cg.info` veranlaßt den Coder ein File zu erzeugen, das die Codierung der Videosequenz protokolliert:

```
Frame: 0
Bits per frame: 304128
Lambda:         ---
R-D:            304128.000000 0.000000
PSNR Y:         NaN
```

```
PSNR U:          NaN
PSNR V:          NaN
Frame: 4
Bits per frame: 4430
Lambda:          100.000000
R-D-J:           4430.000000 1018472.000000 1465633.000000
PSNR Y:          32.131496
PSNR U:          48.866229
PSNR V:          49.793113
```

Für jedes Bild wird die Bildnummer, Datenrate, Distortion und die Lagrange-Kosten festgehalten. Die Distortion der Farbkomponenten für ein Bild ist der *quadratische Fehler*

$$\begin{aligned} D_Y[k] &= \sum_{m=1}^{M}\sum_{n=1}^{N} |\mathbf{x}_Y[m,n,k] - \mathbf{y}_Y[m,n,k]|^2 \\ D_{U,V}[k] &= \sum_{m=1}^{\frac{M}{2}}\sum_{n=1}^{\frac{N}{2}} |\mathbf{x}_{U,V}[m,n,k] - \mathbf{y}_{U,V}[m,n,k]|^2 \end{aligned}$$

und die Gesamtdistortion berechnet sich aus der Summe

$$D[k] = D_Y[k] + D_U[k] + D_V[k].$$

Die PSNR-Werte der Farbkomponenten erhält man aus den partiellen Distortion-Werten:

$$PSNR_Y[k] = 10\log_{10}\left(\frac{255^2}{\frac{1}{MN}D_Y[k]}\right) \quad PSNR_{U,V}[k] = 10\log_{10}\left(\frac{255^2}{\frac{4}{MN}D_{U,V}[k]}\right)$$

Die Lagrange-Kosten J ergeben sich *nicht* direkt aus $D + \lambda R$, da

1. λ für die Farbkomponenten modifiziert wird ($\lambda_U = \lambda_V = \frac{1}{4}\lambda_Y$) und
2. die Überlappung der Segmente bei der Bewegungsschätzung nicht berücksichtigt wird.

`-C /POOL_HOME/stud/mflierl/Coding/Quadtree/Codec/tab/itd1.0.cbk` legt das für die Codierung zu verwendende Codebuch fest. Das Codebuch beschreibt die Quantisierung der Abbildung des vorangegangenen Bildes auf das aktuelle Bild:

```
Codebook 1 (CodebookType)

Quadtree Structure 4 3 3 (DepthYUV[])

Shape Byte: 0 (Component), 0 (Level), 16 (Size)
15 11
...
9 1010110

Spatial Displacement: 0 (Component), 0 (Level), 289 (Size)
```

```
0 0 1
1 0 0010
...
8 8 0110110100110100

Intensity Displacement: 0 (Component), 0 (Level), 0 (Size)

Shape Byte: 0 (Component), 1 (Level), 16 (Size)
0 10
...
14 00100111

Spatial Displacement: 0 (Component), 1 (Level), 361 (Size)
0 0 1
-1 0 0110
...
9 9 001100100000001

Intensity Displacement: 0 (Component), 1 (Level), 77 (Size)
-152 1010011001010111
...
-4 11
0 0
4 100
...
152 1010011001010110

Shape Byte: 0 (Component), 2 (Level), 16 (Size)
0 1
...
15 0001101111

Spatial Displacement: 0 (Component), 2 (Level), 961 (Size)
0 0 01
-1 0 0011
...
15 15 1110111100100101

Intensity Displacement: 0 (Component), 2 (Level), 97 (Size)
-192 1100110001010111
...
192 1100110001010110


Shape Byte: 0 (Component), 3 (Level), 0 (Size)

Spatial Displacement: 0 (Component), 3 (Level), 1369 (Size)
0 0 10
-1 0 0111
...
18 18 1110111000010101
```

```
Intensity Displacement: 0 (Component), 3 (Level), 93 (Size)
-184 1010000100000111
...
184 1010000100000110

Shape Byte: 1 (Component), 0 (Level), 16 (Size)
0 0
3 1011
...
6 100110011

Spatial Displacement: 1 (Component), 0 (Level), 0 (Size)

Intensity Displacement: 1 (Component), 0 (Level), 0 (Size)

Shape Byte: 1 (Component), 1 (Level), 16 (Size)
0 0
...
15 1001011111

Spatial Displacement: 1 (Component), 1 (Level), 225 (Size)
1 0 001
-1 0 010
...
7 7 1001110101100100

Intensity Displacement: 1 (Component), 1 (Level), 41 (Size)
-80 1010001000011100
...
80 101000100001111

Shape Byte: 1 (Component), 2 (Level), 0 (Size)

Spatial Displacement: 1 (Component), 2 (Level), 441 (Size)
0 0 1
...
10 10 0011010011101011

Intensity Displacement: 1 (Component), 2 (Level), 53 (Size)
-104 1100111001011111
...
104 1100111001010001


Shape Byte: 2 (Component), 0 (Level), 15 (Size)
0 0
...
9 101010110

Spatial Displacement: 2 (Component), 0 (Level), 0 (Size)
```

```
Intensity Displacement: 2 (Component), 0 (Level), 0 (Size)

Shape Byte: 2 (Component), 1 (Level), 16 (Size)
0 0
...
15 10011100011

Spatial Displacement: 2 (Component), 1 (Level), 225 (Size)
1 0 001
-1 0 010
...
7 7 1001110000001100

Intensity Displacement: 2 (Component), 1 (Level), 41 (Size)
-80 1010000100110011
...
80 1010000100110010

Shape Byte: 2 (Component), 2 (Level), 0 (Size)

Spatial Displacement: 2 (Component), 2 (Level), 441 (Size)
0 0 1
0 -1 00100
...
10 10 0101001101000101

Intensity Displacement: 2 (Component), 2 (Level), 43 (Size)
-84 1100101110001001
...
84 110010111001110
```

Die Beschreibung der Quadtree-Struktur ist zwingend notwendig. `Quadtree Structure 4 3 3 (DepthYUV[])` definiert für die Y-Komponente einen allgemeinen Quadtree mit vier Ebenen, für die U- und V-Komponente einen mit jeweils drei Ebenen. Eine Ebene d in der Komponente c ist durch den Marker `c (Component), d (Level)` gekenzeichnet. Ein partielles Codebuch ist ohne Ausnahme durch `Shape Byte:`, `Spatial Displacement:` und `Intensity Displacement:` strukturiert. Ist für einen Parameter des partiellen Codebuchs eine Codierung nicht notwendig, so wird der Marker `0 (Size)` verwendet, der die Anzahl der Codeworte in der Tabelle auf Null setzt. So ist z.B. jeweils auf dem tiefsten Level die Größe der Shape-Tabelle und auf dem Top-Level jeder Komponente die Größe der Tabelle des Intensitätsausgleichs auf Null gesetzt. Um die Codierungsgeschwindigkeit zu maximieren, sind die Einträge der `Shape`- und `Spatial Displacement`-Tabelle nach ihren Auftrittswahrscheinlichkeiten sortiert. Bei der Codierung werden somit häufige Codewort-Kombinationen zuerst gepfüft. Eine Lattice-Quantisierung der Intensitätsdifferenz vermeidet eine vollständige Suche in der `Intensity Displacement`-Tabelle, wodurch diese nach Differenzwerten strukturiert ist.

`-O 1` erlaubt eine überlappende Bewegungskompensation der Segmente und reduziert somit die auftretenden Blockeffekte. (In der Version 1.0 ist die überlappende Bewe-

gungsschätzung nicht implementiert.) Die Gewichtungsmasken für die *Overlapped Motion and Intensity Compensation* werden durch die Option `-K /POOL_HOME/stud/mflierl/Coding/Quadtree/Codec/tab/mask.tab` spezifiziert.

```
mask for 5 levels

18 size 0 v 0 h
...

10 size 0 v 0 h
...

6 size 0 v 0 h
...

4 size 0 v 0 h
...

3 size 0 v 0 h
2 3 2
3 5 3
2 3 2

3 size 0 v 1 h
0 0 0
5 8 5
2 3 2

3 size 0 v -1 h
2 3 2
5 8 5
0 0 0

3 size 1 v 0 h
0 5 2
0 8 3
0 5 2

3 size 1 v 1 h
0 0 0
0 13 5
0 5 2

3 size 1 v -1 h
0 5 2
0 13 5
0 0 0

3 size -1 v 0 h
2 5 0
```

```
3 8 0
2 5 0

3 size -1 v 1 h
0 0 0
5 13 0
2 5 0

3 size -1 v -1 h
2 5 0
5 13 0
0 0 0
```

Im Rahmen dieser Arbeit wurden Blöcke der Größe 16×16, 8×8, 4×4, 2×2, 1×1 zur Kompensation herangezogen, da die auftretenden Segmente mit einer maximalen Kantenlänge von 32 Pixel in diese Blockgrößen zerlegt werden können. Da die Überlappung um eine Pixelbreite an den Rändern gesondert betrachtet werden muß, sind weitere acht Randmasken für jede Blockgröße spezifiziert. In dem Auszug aus diesem File werden die neun Masken zur Gewichtung eines 1x1 Blocks dargestellt. Die Masken werden in auftretender Reihenfolge für folgende Postionen verwendet: innerer Bereich, obere, untere, linke, obere linke, untere linke, rechte, obere rechte und untere rechte Randposition. Die Masken für größere Blöcke ergeben sich durch Überlagerung der Masken für 1×1 Blöcke.

A.1.2 Decoder

Der VBS-Decoder generiert aus der gegebenen Codesequenz die Rekonstruktion der ursprünglichen Videosequenz. Der Decoder arbeitet mit folgenden Optionen:

```
VBS coder version 1.0
(C) 1996 University of Erlangen-Nuremberg
    T. Wiegand, M. Flierl

QtDecode [-<help>hio<orig><info>CK<ff><RFF>nmabsfkDzMdO<sc>]
    -help        verbose help
    -h           help
    -i    [name] code file name
                 [bits.itd]
    -o    [name] reconstructed sequence file name
                 []
    -orig [name] original sequence file name
                 []
    -info [name] information file name
    -C    [name] codebook file name
                 [/POOL_HOME/stud/mflierl/Coding/Quadtree/Codec/tab/itd1.0.cbk]
    -K    [name] mask file name
                 [/POOL_HOME/stud/mflierl/Coding/Quadtree/Codec/tab/mask.tab]
    -ff   [name] first frame file name
```

```
-RFF  [num]  bits used for first frame [0]
-n    [num]  input image width  [176]
-m    [num]  input image height [144]
-a    [num]  start              [0]
-b    [num]  end                [99]
-s    [num]  frame step         [1]
-f    [num]  frequency          [30]
-k    [0/1]  khoros format      [0]
-D    [num]  output mode        [2]
               0: write imposed reconstruction file
               1: write reconstruction file
               2: display reconstruction
               3: display imposed reconstruction
-z    [num]  zoom factor        [1]
-M    [num]  macroblock size    [16]
-d    [0/1]  differencing       [1]
-O    [0/1]  overlapping        [1]
-sc   [num]  MV scale factor    [1]
```

Die Option `-i cg.itd` spezifiziert die zu decodierende Codesequenz `cg.itd`, die mit `-o cg_decoded/` in das bereits angelegte Verzeichnis `cg_decoded/` geschrieben wird. Die Referenz `-orig /DATA/sequences/QCIF_CD/mpeg4/coastgua/cg` auf die originale Videosequenz ermöglicht dem Decoder die Berechnung der Distortion zwischen Original und Rekonstruktion. Es wurde bereits im vorigen Abschnitt darauf hingewiesen, daß die Codesequenz keine Steuerinformationen enthält. Aus diesem Grund ist der gleiche Parametersatz für Codierung und Decodierung zu verwenden. Dies gilt insbesondere für das Startbild.

Die Option `-D` legt den Ausgabemodus des Decoders fest. `0` speichert die rekonstruierten Bilder in das mit `-o cg_decoded/` spezifizierte Verzeichnis ab. `1` überlagert zusätzlich die Quadtree-Struktur der Y-Komponente. `2` (`3`) visualisiert die decodierten Bilder (mit überlagerter Quadtree-Struktur der Y-Komponente) am Display, wobei diese noch mit `-z` vergrößert dargestellt werden können.

Mit -info cg_decoded.info wird die Decodierung protokolliert. Das Protokoll des Decoders ist ähnlich dem des Coders:

```
Frame: 0
Bits per frame: 304128
R-D:            304128.000000 0.000000
PSNR Y:         NaN
PSNR U:         NaN
PSNR V:         NaN
Frame: 4
Bits per frame: 4430
R-D:            4430.000000 1018472.000000
PSNR Y:         32.131496
PSNR U:         48.866229
PSNR V:         49.793113
```

Die Option `-sc` soll eine skalierbare Decodierung der Codesequenz ermöglichen. In der Version 1.0 ist damit aber keine funktionale Bedeutung verknüpft.

A.2 Module zur Optimierung des Codebuchs

Das Codebuch besteht aus den Repräsentanten und den ihnen zugeordneten Huffman-Codewörtern variabler Länge. Zur Optimierung des Codebuchs auf der Basis einer gegebenen Trainingsmenge von Videosequenzen wird ein iterativer Algorithmus verwendet, der in jedem Schritt die Lagrange-Kosten verringert und gegen das lokal optimale Codebuch konvergiert.

Um diesen iterativen Algorithmus durchzuführen, wurden Module zur Erzeugung des Ausgungscodebuchs, Bestimmung der Häufigkeitsverteilung und Erzeugung des verbesserten Codebuchs aus der Häufigkeitsverteilung realisiert. Diese werden im folgenden erläutert.

A.2.1 Erzeugung des Ausgangscodebuchs

Für das Ausgangscodebuch werden die Repräsentanten der örtlichen Verschiebung und des Intensitätsausgleichs auf einem Gitter positioniert und einem **F***ix* **L***ength* **C***ode* zugeordnet.

```
InitialCB [-help h C Y U V]
    -help        verbose help
    -h           help
    -C    [name] codebook file name
                 [uniform.cbk]
    -Y    [no no no ..] list of Y codebook parameters of size 'no'
                 [12 16 1089 0 16 1369 77 16 961 97 0 1369 93 ]
    -U    [no no no ..] list of U codebook parameters of size 'no'
                 [9 16 0 0 16 225 41 0 441 53 ]
    -V    [no no no ..] list of V codebook parameters of size 'no'
                 [9 16 0 0 16 225 41 0 441 43 ]
```

Für jeden Level ist ein 3-er Tupel notwendig, das die Anzahl der Einträge in die `Shape`-, `Spatial Displacement`- und `Intensity Displacement`-Tabelle festlegt.

Level	`Shape`	`Spatial Displacement`	`Intensity Displacement`
0	16	1089	0
1	16	1369	77
2	16	961	97
3	0	1369	93

Tabelle A.1: *Dimensionierung der partiellen Codebücher der Y-Komponente*

Ein Codebuch, das vier Ebenen für die Y-Komponente enthält und partielle Codebuchdimensionen wie in **Tabelle A.1** aufweisen soll, wird durch `-Y 12 16 1089 0 16 1369 77`

`16 961 97 0 1369 93` spezifiziert. Der erste Eintrag in die Liste entspricht der Anzahl der folgenden Einträge. Analog sind die U- und V-Komponenten mit den Optionen `-U` und `-V` zu bestimmen. `-C uniform.cbk` definiert den Filenamen des Ausgangscodebuchs.

```
Codebook 0 (CodebookType)

Quadtree Structure 4 3 3 (DepthYUV[])

Shape Byte: 0 (Component), 0 (Level), 16 (Size)
0 1111
1 1110
2 1101
3 1100
4 1011
5 1010
6 1001
7 1000
8 0111
9 0110
10 0101
11 0100
12 0011
13 0010
14 0001
15 00001

Spatial Displacement: 0 (Component), 0 (Level), 1089 (Size)
-16 -16 1111111111
-16 -15 1111111110
-16 -14 1111111101
...
```

Für die Größe der `Shape`-Tabellen sind 2 oder 16 Einträge sinnvoll. Im ersten Fall wird in die Tabelle die Segmentform 0 und 15 eingetragen und entspricht somit einem gewöhnlichen Quadtree, bei dem nur eine *Split*-Möglichkeit zulässig ist. Im zweiten Fall werden alle 15 Möglichkeiten eines *Splits* codiert.

Für die örtliche Verschiebung sind die Positionen eines zweidimensionalen Integer-Lattice $\{-r_s, \ldots, -1, 0, 1, \ldots, r_s\} \times \{-r_s, \ldots, -1, 0, 1, \ldots, r_s\}$ zulässig und die AnzahlN_s der Einträge in die `Spatial Displacement`-Tabelle erhält man aus

$$N_s = (2r_s + 1)^2$$

Ein eindimensionales Integer-Lattice $\{-r_q, \ldots, -4, 0, 4, \ldots, r_q\}$ wird die für die Quantisierung des Intensitätsausgleichs verwendet, wobei sich die Anzahl N_q der Einträge in die `Intensity Displacement`-Tabelle aus

$$N_q = \frac{r_q}{2} + 1$$

ergibt.

A.2.2 Bestimmung der Häufigkeitsverteilung

Für den iterativen Algorithmus ist für das gegebene Trainingsset und Codebuch $\mathcal{B}_i$ die Häufigkeiten der einzelnen Codewörter im Codebuch $\mathcal{B}_i$ zu bestimmen um daraus das verbesserte Codebuch $\mathcal{B}_{i+1}$ zu erhalten. Diese Häufigkeiten werden nun vom Modul `Histogram` bestimmt.

```
Histogram [-<help>hi<ff>ot<info>CK<RFF>nmabskSMdOGlH]
    -help        verbose help
    -h           help
    -i    [name] original sequence file name
                 []
    -ff   [name] first frame file name
    -o    [name] code file name
                 [bits.itd]
    -t    [name] trace file name
    -info [name] information file name
                 [histogram.info]
    -C    [name] codebook file name
                 [/POOL_HOME/stud/mflierl/Coding/Quadtree/Codec/tab/itd1.0.cbk]
    -K    [name] mask file name
                 [/POOL_HOME/stud/mflierl/Coding/Quadtree/Codec/tab/mask.tab]
    -RFF  [num]  bits for first frame    [0 bits]
    -n    [num]  input image width       [176]
    -m    [num]  input image height      [144]
    -a    [num]  start                   [0]
    -b    [num]  end                     [99]
    -s    [num]  frame step              [1]
    -k    [0/1]  khoros format           [0]
    -S    [num]  code buffer size (byte) [100]
    -M    [num]  macroblock size         [16]
    -d    [0/1]  differencing            [1]
    -O    [0/1]  overlapping             [1]
    -G           use growing algorithm
    -l    [no num num ..]   list of lambda values of size 'no'
                 [1 100.000000 ]
    -H    [name] histogram file name
                 [a.hst]
```

Die meisten Optionen sind identisch mit denen des `QtEncode`. Zusätzlich werden aber die Häufigkeiten der Codeworte bei der Codierung der Videosequenz mit dem Codebuch `-C 0.cbk` gezählt und können in dem File `a.hst` mit der Option `-H a.hst` gespeichert werden. Das Protokoll dieses Moduls enthält für jedes Bild eine Zeile, in der λ, Rate, Distortion (quadratische Fehler) und Lagrange-Kosten festgehalten werden.

```
Frame   0
100.00   30754.00  2844938.00  3254277.00
Frame   4
```

```
100.00    8687.00  1937649.00  2737990.00
Frame   8
100.00    5383.00  2011059.00  2509209.00
Frame  12
100.00    5766.00  1981416.00  2500045.00
...
```

Das Histogram-File hat dieselbe Struktur wie das Codebuch-File mit dem Unterschied, daß die Codeworte durch ihre Auftrittshäufigkeiten ersetzt werden:

```
Histogram Generated With 0 (CodebookType)

Quadtree Structure 4 3 3 (DepthYUV[])

Shape Byte: 0 (Component), 0 (Level), 16 (Size)
0 852
1 313
2 295
3 204
4 229
5 560
6 105
7 341
8 235
9 123
10 309
11 223
12 216
13 292
14 181
15 663

Spatial Displacement: 0 (Component), 0 (Level), 1089 (Size)
-16 -16 2
-16 -15 2
-16 -14 3
...
```

Mit der Option `-l 2 100 150` wird die Videosequenz mit $\lambda = 100$ und $\lambda = 150$ codiert, wobei die Codesequenzen in den Files `bits.itd00` und `bits.itd01` gespeichert werden. Die Zahl nach der Option `-l` gibt die Anzahl der folgenden λ-Werte an und die Codesequenzen werden mit Null beginnend abgezählt und gekennzeichnet.

A.2.3 Erzeugung des Codebuchs aus der Häufigkeitsverteilung

Das Modul `UpdateCodebook` erlaubt es, mehrere Häufigkeitsverteilungen zu akkumulieren und daraus einen Huffman-Code zu generieren.

```
UpdateCodebook [-help h info C NewC K H]
    -help        verbose help
    -h           help
    -info [name] information file name
                 [update.info]
    -C    [name] codebook file name
                 [old.cbk]
    -NewC [name] file name of new codebook
                 [new.cbk]
    -K    [name] mask file name
                 [/POOL_HOME/stud/mflierl/Coding/Quadtree/Codec/tab/mask.tab]
    -H    [no name name ..] list of histogram file names of size 'no'
                 [1 a.hst ]
    -S    [name] accumulated histogram file name
                 [sum.hst]
```

Das aktuelle Codebuch, mit dem die Häufigkeitsverteilungen erzeugt wurden, ist mit der Option `-C old.cbk` zu spezifizieren. -H 3 a.hst b.hst c.hst legt die `Histogram`-Files fest, wobei die Zahl nach der Option `-H` die Anzahl der folgenden Filenamen bestimmt. `-S sum.hst` speichert die akkumulierten Häufigkeitsverteilungen in `sum.hst`.

`-info update.info` protokolliert den Ablauf des Moduls `UpdateCodebook`. Das Protokoll gliedert sich dabei in folgende Punkte:

1. einlesen und akkumulieren der Codewort-Häufigkeiten,
2. aussortieren der unbenutzten Codeworte,
3. erzeugen der neuen Huffman-Tabellen und
4. sortieren der Codeworte nach ihren Längen.

```
Reading histogram file: ak.hst
Reading histogram file: bm.hst
Reading histogram file: cg.hst
Reading histogram file: ch.hst
Reading histogram file: cs.hst
Reading histogram file: fl.hst
Reading histogram file: fm.hst
Reading histogram file: hm.hst
Reading histogram file: mc.hst
Reading histogram file: md.hst
Reading histogram file: nw.hst
Reading histogram file: se.hst
Reading histogram file: si.hst
Reading histogram file: st.hst
Reading histogram file: td.hst
Reading histogram file: tl.hst
Reading histogram file: tt.hst
Reading histogram file: we.hst
```

```
Wrote accumulated histogram file: sum.hst

Generate shape codebook (component 0, level 0): o.k!
Generate spatial displacement codebook (component 0, level 0): o.k!
Generate intensity displacement codebook (component 0, level 0): o.k!
Generate shape codebook (component 0, level 1): o.k!
Generate spatial displacement codebook (component 0, level 1): o.k!
Generate intensity displacement codebook (component 0, level 1): o.k!
Generate shape codebook (component 0, level 2): o.k!
Generate spatial displacement codebook (component 0, level 2): o.k!
Generate intensity displacement codebook (component 0, level 2): o.k!
Generate shape codebook (component 0, level 3): o.k!
Generate spatial displacement codebook (component 0, level 3): o.k!
Generate intensity displacement codebook (component 0, level 3): o.k!
Generate shape codebook (component 1, level 0): o.k!
Generate spatial displacement codebook (component 1, level 0): o.k!
Generate intensity displacement codebook (component 1, level 0): o.k!
Generate shape codebook (component 1, level 1): o.k!
Generate spatial displacement codebook (component 1, level 1): o.k!
Generate intensity displacement codebook (component 1, level 1): o.k!
Generate shape codebook (component 1, level 2): o.k!
Generate spatial displacement codebook (component 1, level 2): o.k!
Generate intensity displacement codebook (component 1, level 2): o.k!
Generate shape codebook (component 2, level 0): o.k!
Generate spatial displacement codebook (component 2, level 0): o.k!
Generate intensity displacement codebook (component 2, level 0): o.k!
Generate shape codebook (component 2, level 1): o.k!
Generate spatial displacement codebook (component 2, level 1): o.k!
Generate intensity displacement codebook (component 2, level 1): o.k!
Generate shape codebook (component 2, level 2): o.k!
Generate spatial displacement codebook (component 2, level 2): o.k!
Generate intensity displacement codebook (component 2, level 2): o.k!

New codebook sorted:
  Page size reduced: 1089 --> 1084 (spatial displacement table, component 0,
                                    level 0)
```

Ein Ausschnitt des verbesserten Codebuchs macht die Sortierung und die Reduktion der Anzahl der Codeworte deutlich.

```
Codebook 1 (CodebookType)

Quadtree Structure 4 3 3 (DepthYUV[])

Shape Byte: 0 (Component), 0 (Level), 16 (Size)
15 11
0 101
5 0110
```

```
3 1000
8 0101
2 1001
10 0100
4 0111
12 0011
7 00100
11 00011
1 00101
14 00001
13 00010
6 000001
9 0000001

Spatial Displacement: 0 (Component), 0 (Level), 1084 (Size)
0 0 11
1 0 1010
-1 0 1011
...
```

A.2.4 Sortierung des Codebuchs

Ist bereits ein Codebuch im Fileformat nach **Abschnitt A.1.1** vorhanden, dessen `Shape`- und `Spatial Dislacement`-Tabellen aber nicht nach den Codewortlängen sortiert sind, so kann mit dem Modul `SortCodebook` eine Erhöhung der Codierungsgeschwindigkeit erreicht werden.

```
SortCodebook [-help h C NewC]
    -help        verbose help
    -h           help
    -C    [name] codebook file name
                 [unsorted.cbk]
    -NewC [name] file name of new codebook
                 [sorted.cbk]
```

`-C unsorted.cbk` spezifiziert das unsortierte Codebuch, `-NewC sorted.cbk` definiert den Filenamen des sortierten Codebuchs. Ausschnitte aus einem unsortierten und dem ensprechenden sortierten Codebuch verdeutlichen die Funktionsweise des Moduls `SortCodebook`.

```
Codebook 1 (CodebookType)

Quadtree Structure 4 3 3 (DepthYUV[])

Shape Byte: 0 (Component), 0 (Level), 16 (Size)
0 000
1 0110
2 1000
```

```
3 0101
4 1001
5 0010
6 101010
7 00110
8 1011
9 1010110
10 0111
11 00111
12 0100
13 1010111
14 10100
15 11

Spatial Displacement: 0 (Component), 0 (Level), 289 (Size)
-8 -8 0110110100110111
-8 -7 01011100010010
-8 -6 0111101001110
...




Codebook 1 (CodebookType)

Quadtree Structure 4 3 3 (DepthYUV[])

Shape Byte: 0 (Component), 0 (Level), 16 (Size)
15 11
0 000
4 1001
2 1000
3 0101
8 1011
5 0010
10 0111
1 0110
12 0100
7 00110
11 00111
14 10100
6 101010
13 1010111
9 1010110

Spatial Displacement: 0 (Component), 0 (Level), 289 (Size)
0 0 1
1 0 0010
-1 0 0011
...
```

A.3 Analyse-Modul

Das Analyse-Modul `Freq` extrahiert verschiedene Daten aus einer gegebenen Codesequenz. Es basiert auf dem Modul `QtDecode` und die möglichen Optionen entsprechen deshalb denen des Decoders.

```
VBS coder version 1.0
(C) 1996 University of Erlangen-Nuremberg
    T. Wiegand, M. Flierl

Freq [-<help>hio<orig><info>CK<ff><RFF>nmabsfkDzMdO<sc>]
    -help        verbose help
    -h           help
    -i    [name] code file name
                 [bits.itd]
    -o    [name] reconstructed sequence file name
                 []
    -orig [name] original sequence file name
                 []
    -info [name] information file name
    -C    [name] codebook file name
                 [/POOL_HOME/stud/mflierl/Coding/Quadtree/Codec/tab/itd1.0.cbk]
    -K    [name] mask file name
                 [/POOL_HOME/stud/mflierl/Coding/Quadtree/Codec/tab/mask.tab]
    -H    [name] histogram file name
                 [out.hst]
    -ff   [name] first frame file name
    -RFF  [num]  bits used for first frame [0]
    -n    [num]  input image width  [176]
    -m    [num]  input image height [144]
    -a    [num]  start              [0]
    -b    [num]  end                [99]
    -s    [num]  frame step         [1]
    -f    [num]  frequency          [30]
    -k    [0/1]  khoros format      [0]
    -D    [num]  output mode        [2]
                   0: write imposed reconstruction file
                   1: write reconstruction file
                   2: display reconstruction
                   3: display imposed reconstruction
    -z    [num]  zoom factor        [1]
    -M    [num]  macroblock size    [16]
    -d    [0/1]  differencing       [1]
    -O    [0/1]  overlapping        [1]
    -sc   [num]  MV scale factor    [1]
```

`-info freq.info` spezifiziert das Informationsfile, das Daten der decodierten Codesequenz protokolliert. Für jedes Bild wird die Bildnummer, Datenrate und Distortion festgehalten. Am Ende des Files werden die partiellen Raten der decodierten Bilder festge-

halten. Für jede Komponente und Ebene sind die Raten für die Segmentformen, örtliche Verschiebungen und Intensitätsdifferenzen tabelliert.

```
Frame: 0
Bits per frame: 304128
R-D:            304128.000000 0.000000
PSNR Y:         NaN
PSNR U:         NaN
PSNR V:         NaN
Frame: 4
Bits per frame: 4430
R-D:            4430.000000 1018472.000000
PSNR Y:         32.131496
PSNR U:         48.866229
PSNR V:         49.793113

FF-Rate:        304128.000000
Partial Rates for Encoded Frames:
Skip-Rate       99.000000
Partial Rates (Component) (Level):
S-XY-Q-Rate 0 0 208.000000      158.000000      0.000000
S-XY-Q-Rate 0 1 470.000000      831.000000      197.000000
S-XY-Q-Rate 0 2 306.000000      1109.000000     422.000000
S-XY-Q-Rate 0 3 0.000000        277.000000      179.000000
S-XY-Q-Rate 1 0 70.000000       0.000000        0.000000
S-XY-Q-Rate 1 1 6.000000        22.000000       6.000000
S-XY-Q-Rate 1 2 0.000000        0.000000        0.000000
S-XY-Q-Rate 2 0 65.000000       0.000000        0.000000
S-XY-Q-Rate 2 1 1.000000        3.000000        1.000000
S-XY-Q-Rate 2 2 0.000000        0.000000        0.000000
```

`-H freq.hst` bestimmt das `Histogram`-File, das die Häufigkeiten der einzelnen Codeworte für die decodierte Codesequenz beinhaltet.

```
Histogram Generated With 1 (CodebookType)

Quadtree Structure 4 3 3 (DepthYUV[])

Shape Byte: 0 (Component), 0 (Level), 16 (Size)
15 20
0 4
4 3
2 2
3 2
8 4
5 3
10 10
1 3
12 5
```

```
7 1
11 2
14 0
6 1
13 0
9 1

Spatial Displacement: 0 (Component), 0 (Level), 289 (Size)
0 0 14
1 0 7
-1 0 3
...
```

Anhang B

Beschreibung der Objekte

Die in *Abschnitt A* beschriebenen Module sind aus zahlreichen Objekten [25] zusammengesetzt. Objekte, die nicht in direktem Zusammenhang mit dem *Variable Block Size Codec* stehen und für andere Anwendungen benutzt werden können, sind in einer Bibliothek zusammengefaßt. In diesem Kapitel soll nun versucht werden, die verwendeten Objekte anschaulich zu beschreiben.

B.1 Bibliotheksobjekte

Die Bibliothek besteht aus Objekten zur Beschreibung von Blöcken variabler Größe innerhalb eines Bildes, zur Huffman-Codierung und zur Strukturierung von Daten in Baumstrukturen.

B.1.1 Block-Beschreibung mit `Block`

Um Blöcke variabler Größe (N, M) eines Bildes effektiv handzuhaben, wurde eine Blockbeschreibung definiert.

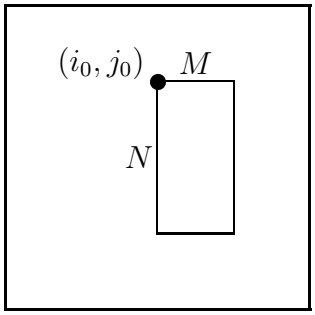


Abbildung B.1: *Blockbeschreibung: Zeiger auf einen Block variabler Größe*

Die Blockbeschreibung setzt sich aus einem Blockzeiger (i_0, j_0), der auf die linke obere Ecke des Blocks verweist, und den Größenangaben (N, M) zusammen. **(Siehe Abbildung B.1.)** Die in dem Objekt definierten Methoden [25] benötigen die Daten des Bildes

und die Blockbeschreibung, da diese keine Bilddaten enthält. Die Vorteile dieser Blockbeschreibung liegen zum einen beim geringen Speicherbedarf, da die Blockbeschreibung keine mit dem Block assoziierten Daten enthält, und zum anderen bei der effizienten Datenbearbeitung, da die Blockinhalte nicht kopiert werden müssen.

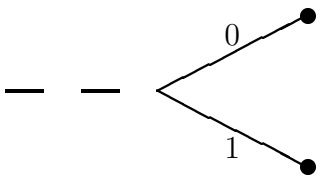
B.1.2 Huffman-Codes mit `Code`

Die Huffman-Codierung stellt eine einfache und optimale Konstruktionsmethode zur Codierung einer ergodischen Markov-Quelle dar. Der Huffman-Algorithmus nach **Abbildung B.2** [13] generiert einen binären präfixfreien Code, der folgende Eigenschaften besitzt:

1. Keine zwei Codeworte dürfen identisch sein.
2. Kein Codewort darf Präfix eines längeren Codewortes sein. Diese Forderung bedeutet, daß man ein Codewort erkennen kann, sobald dessen letztes Symbol empfangen wurde.

Schritt 0: Die k Indizes seien die Endknoten eines binären Baumes mit unbekannter Struktur. Man ordne den Endknoten die Auftrittswahrscheinlichkeiten p_i für $i = 1, 2, \ldots, k$ der Indizes zu. Betrachte diese k Knoten als *aktiv*.

Schritt 1: Vereinige die beiden am wenigsten wahrscheinlichen Knoten mit einem binären Zweig wie folgt:



Setze diese beiden Knoten *passiv*, setze den neugeschaffenen Knoten *aktiv* und weise ihm die Summe der Wahrscheinlichkeiten der beiden verbundenen Knoten zu.

Schritt 2: Wenn nur noch ein aktiver Knoten übrig ist, sind wir an der Wurzel und halten an. Im anderen Fall gehe zu Schritt 1.

Abbildung B.2: *Huffman-Algorithmus zur Konstruktion eines binären präfixfreien Codes*

Das folgende Beispiel [26] verdeutlicht den Huffman-Algorithmus. Es werden fünf Indizes A, B, C, D, E mit den Wahrscheinlichkeiten $p_A = 0.264, p_B = 0.053, p_C = 0.108, p_D = 0.137, p_E = 0.438$ codiert. Die Schritte zur Konstruktion des Huffman-Baums werden in **Abbildung B.3** veranschaulicht:

1. Verbinde B und C zu einem Baum der Wahrscheinlichkeit $p_{BC} = 0.162$. p_{BC} bezeichnet dabei die Wahrscheinlichkeit, daß entweder Index B oder Index C auftritt.
2. Verbinde BC und D zu einem Baum der Wahrscheinlichkeit $p_{BCD} = 0.298$.
3. Verbinde A und BCD zu einem Baum der Wahrscheinlichkeit $p_{ABCD} = 0.562$.
4. Verbinde ABCD und E zu einem Baum der Wahrscheinlichkeit $p_{ABCDE} = 1.000$.

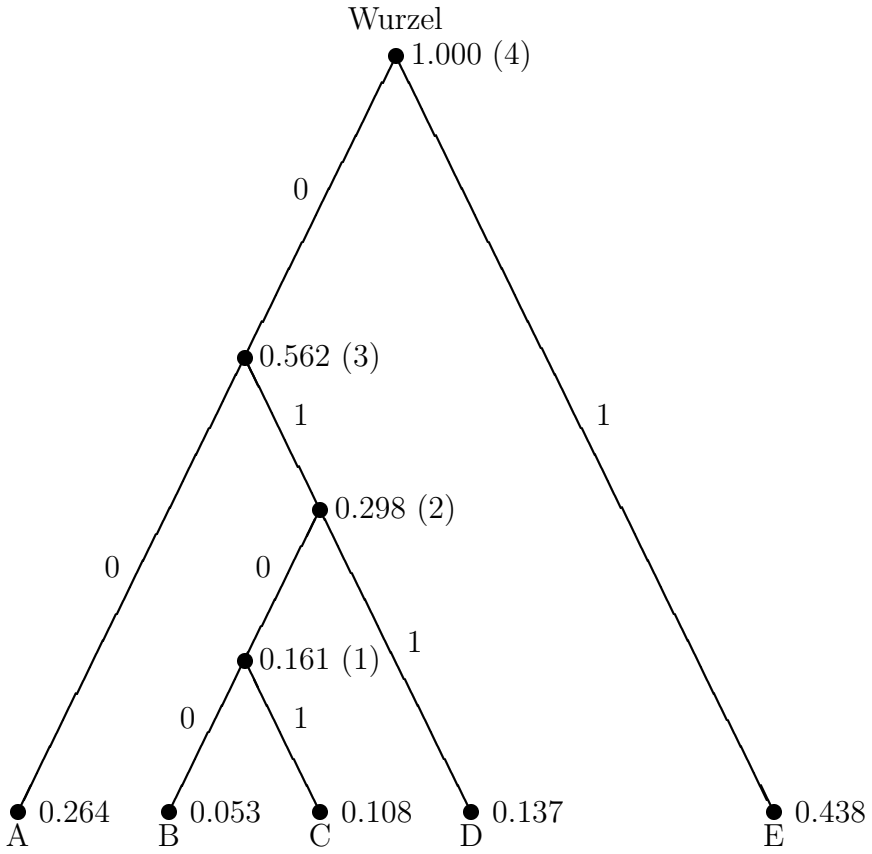


Abbildung B.3: *Konstruktion eines Huffman-Baums*

Index	Wahrscheinlichkeit	Codewort
A	0.264	00
B	0.053	0100
C	0.108	0101
D	0.137	011
E	0.438	1

Tabelle B.1: *Zuordung der Codeworte zu den Indizes*

Tabelle B.1 zeigt die Zuordung der Codeworte zu den Indizes. Zur Beurteilung der Qualität des Huffman-Codes wird abschließend die Index-Entropie H mit der mittleren Codewortlänge L verglichen. Die Index-Entropie ist die untere Schranke für die mittlere Codewortlänge und wird für dieses Beispiel relativ gut erreicht.

$$H = -\sum_{i=1}^{k} p_i \log_2 p_i = 1.993\text{bit} \qquad L = \sum_{i=1}^{k} p_i l_i = 2.021\text{bit}$$

B.1.3 Baumstrukturen mit `Tree`

Die hierarchische Strukturierung von Daten nimmt in dieser Arbeit eine zentrale Position ein. Zum einen werden die bewegungskompensierten Blöcke dieser Art der Strukturierung unterworfen und zum anderen können die verwendeten Codeworte durch einen Huffman-Baum **(Abschnitt B.1.2)** repräsentiert werden. Der binäre Code wird mit dem Objekt `Bintree` und die hierarchische Strukturierung der Segmente mit dem Objekt `Quadtree` realisiert.

Das generische Element eines Baumes ist ein Knoten mit den daran anschließenden Zweigen. Verlassen einen Knoten maximal zwei Zweige, so existieren $2^2 = 4$ mögliche Knotenformen („Zustände“ des Knoten).

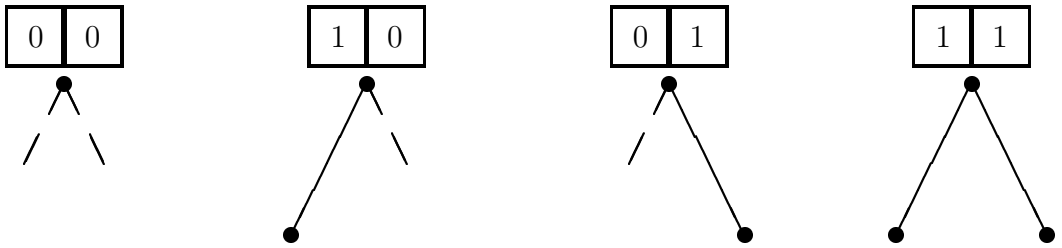


Abbildung B.4: Knotenformen für einen Baum mit bis zu zwei Zweigen pro Knoten

In **Abbildung B.4** sind diese vier Knotenformen dargestellt. Wird ein Zweig verwendet, so wird dieser mit „1“ gekennzeichnet (ausgezogener Zweig). Ungenutzte Zweige (gestrichelte Zweige) werden hingegen mit „0“ markiert. Zusammenfassend ist zu bemerken, daß nicht nur zwischen Endknoten (äußerer Knoten eines Baumes) und innerer Knoten zu unterscheiden ist. Zusätzlich ist zu spezifizieren, ob der linke oder rechte Zweig bzw. beide Zweige den inneren Knoten verlassen.

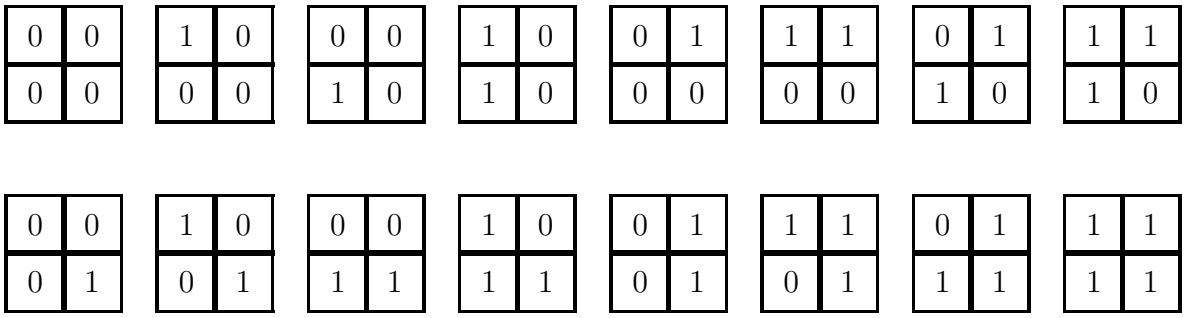


Abbildung B.5: Knotenformen für einen Baum mit bis zu vier Zweigen pro Knoten

Das Objekt `Quadtree` berücksichtigt den Fall, daß maximal vier Zweige einen Knoten verlassen. Somit existieren $2^4 = 16$ mögliche Knotenformen. **Abbildung B.5** zeigt die 16 Knotenformen, wobei jeder Zweig durch einen Quadranten repräsentiert wird. Ist dieser Quadrant mit einer „1“ gekennzeichnet, so existiert ein Zweig, an dem ein weiterer Teilbaum geknüpft ist. „0“ kennzeichnet die Quadranten, die dem Knoten den Charakter

eines Endknotens verleihen. Zusammenfassend läßt sich feststellen, daß diese Beschreibung eines Knotes nicht nur den „Zustand“ eines inneren bzw. äußeren Knoten, sondern auch eine Abstufung zwischen diesen beiden Extremen zuläßt.

B.2 VBS-Codec-Objekte

Objekte von der Spezifikation des Codebuchs bis zur Stukturbeschreibung des Codecs sind notwendig, um Coder und Decoder zu implementieren, den iterativen Algorithmus zu realisieren und Vergleiche zwischen verschiedenen Bitzuweisungsstrategien anzustellen. Dieser Abschnitt dient zur Beschreibung der notwendigen Objekte.

B.2.1 `Codebook`

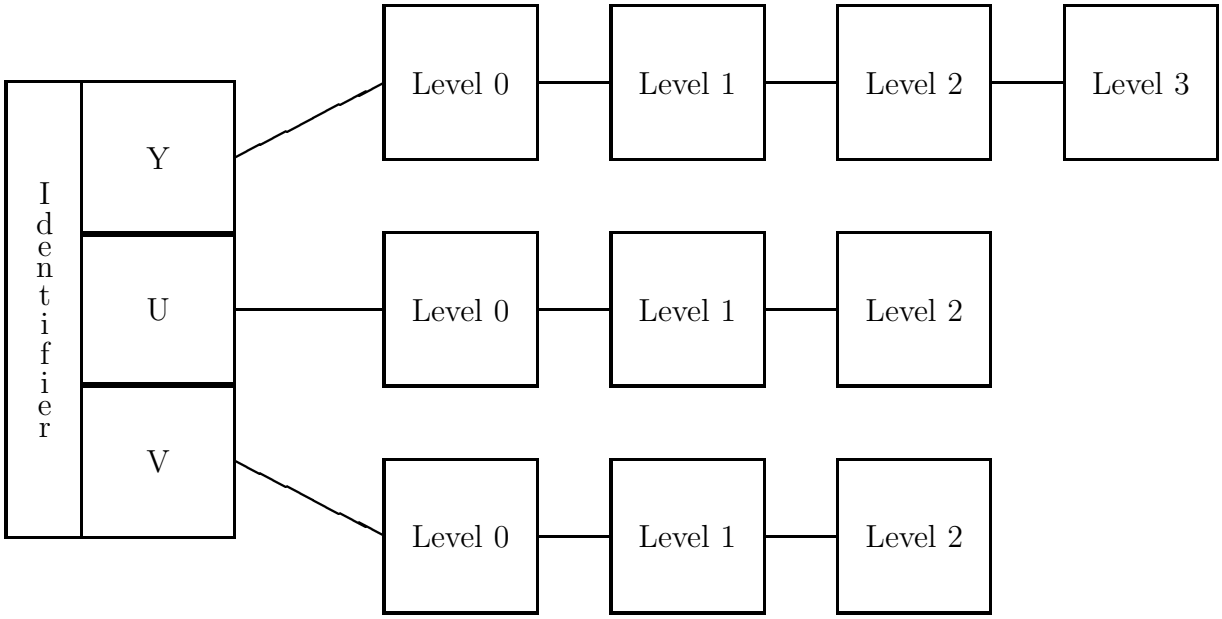


Abbildung B.6: Struktur des vollständigen Codebuchs

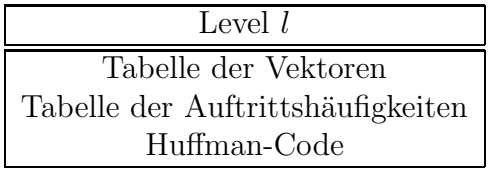


Abbildung B.7: *Struktur des partiellen Codebuchs auf dem Level l*

Die Zuordnung zwischen Vektoren, Häufigkeiten und Codeworten für alle Farbkomponenten und Hierarchiestufen wird durch das Objekt `Codebook` ermöglicht. Auf jedem Level l und jeder Farbkomponente beinhaltet ein partielles Codebuch eine Tabelle der Vektoren, eine Häufigkeitstabelle und einen Huffman-Code (**Abbildung B.7**). Um nach Farbkomponenten und Hierarchiestufen zu gliedern, wurde das vollständige Codebuch nach **Abbildung B.6** strukturiert.

B.2.2 `Codec`

Dieses Objekt enthält neben dem Coder auch den Decoder und trägt somit der Symmetrie zwischen beiden Bausteinen Rechnung. Diese Kombination ermöglicht auch Definitionen von Methoden (typspezifische Funktionen eines Objekts [25]), die von beiden Bausteinen benutzt werden können (z.B. Initialisierung, Überlagerung der Quadtree-Struktur auf das jeweilige Bild).

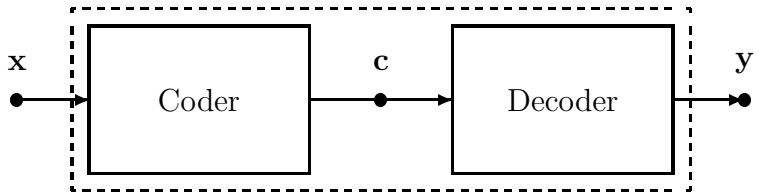


Abbildung B.8: *Struktur des Codec*

Das Zeitverhalten des Codec läßt sich durch eine Feedback-Struktur wie in **Abbildung B.9** beschreiben. Die Rekonstruktion des Bildes $\mathbf{y}[k]$ zur diskreten Zeit k ist nicht nur vom Original $\mathbf{x}[k]$, sondern auch von der Rekonstruktion $\mathbf{y}[k-1]$ des vorherigen Bildes abhängig. Somit kann die Bildsequenz am Decoder nur dann eindeutig rekonstruiert werden, wenn das Startbild am Coder und Decoder identisch ist.

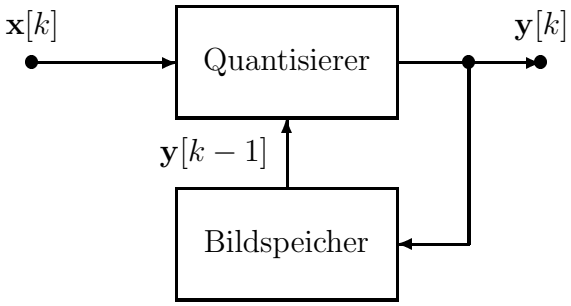


Abbildung B.9: *Feedback-Struktur des Codec*

B.2.3 `Codeio`

Die Parameter des Modells wie Segmentform, Bewegungsvektor und Intensitätsdifferenz werden codiert. Jedem Parameter wird je nach Häufigkeit des Auftretens ein Codewort variabler Länge (VLC) zugeordnet.

Abbildung B.10 erläutert die Funktionalität des Objektes `Codeio`. Die Indexsequenz **i** wird durch den Baustein α geschätzt. Dieser wird im Baustein γ ein Kanalcodesequenz **c** zugeordnet, das aus einer VLC-Tabelle ausgelesen wird. Da diese Zuordung eineindeutig und keine Störung auf dem Kanal (speichern und lesen der Codesequenz) zugelassen ist, kann der Baustein γ^{-1} die Indexsequenz eindeutig aus dem Kanalcodewort wiedergewinnen. Der Baustein β ist für die Kompensation der Indexsequenz zuständig und generiert daraus die Rekonstruktion.

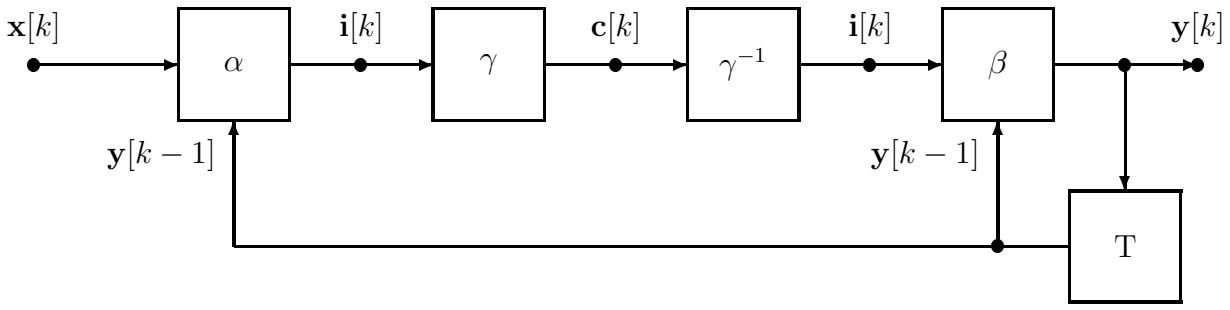


Abbildung B.10: *Entropie-Codierung der Modellparameter*

B.2.4 `ECVQ`

Dieses Objekt realisiert den iterativen Algorithmus. Die Schätzung der Modellparameter und die Zuordung von Huffman-Codeworten beeinflussen sich wechselseitig. Für die Schätzung eines Modellparameters ist die Länge des Huffman-Codewortes notwendig und für die Generierung der Huffman-Codeworte ist die Häufigkeitsverteilung der Modellparameter anzugeben. Da somit eine unabhängige Optimierung der Bausteine (α, β) und γ (**siehe Abbildung B.10**) nicht möglich ist, werden alle Bausteine durch eine iterativen Algorithmus nach **Abbildung B.11** [15] optimiert.

Schritt 0: Setze $\kappa = 0$ und initialisiere den Codec $(\alpha^{(0)}, \beta^{(0)}, \gamma^{(0)})$

Schritt 1: Schätze und kompensierte die Modellparameter optimal mit der Entropie-Codierung $\gamma^{(\kappa)}$ und bestimme die dazu gehörige Häufigkeitsverteilung der mit $(\alpha^{(\kappa)}, \beta^{(\kappa)})$ geschätzten bzw. kompensierten Modellparameter.

Schritt 2: Bestimme aus der Häufigkeitsverteilung die verbesserte Entropie-Codierung $\gamma^{(\kappa+1)}$.

Schritt 3: Solange eine Verbesserung möglich ist, erhöhe $\kappa := \kappa + 1$ und fahre mit Schritt 1 fort.

Abbildung B.11: *Iterativer Algorithmus zur Optimierung des Codecs*

Schritt 1 und 2 werden in diesem Objekt realisiert. Die Funktion `Histogram` schätzt und kompensiert die Modellparameter und akkumuliert die Häufigkeiten für jedes Bild. Die Funktion `UpdateCodebook` erzeugt aus der Häufigkeitsverteilung den verbesserten Entropie-Code.

B.2.5 `Estimation`

In diesem Objekt werden drei wichtige Probleme gelöst. Die Optimierung ohne Nebenbedingung, die Bestimmung der optimalen Intensitätsdifferenz und die effektive Suche im Parameterraum werden an dieser Stelle kurz erläutert.

Die optimale Indexsequenz $\hat{\mathbf{i}}$ des Modells wird mit Hilfe der Methode der Lagrange-Multiplikatoren bestimmt [14]. Diese Methode ermöglicht eine Umformulierung eines Optimierungsproblems mit Nebenbedingung (in diesem Fall die Rate) zu einem ohne Nebenbedinung. Um die optimale Bitzuweisung, d.h den optimalen Parametersatz zu determinieren, werden die Lagrange-Kosten minimiert.

$$\hat{J} = J(\hat{\mathbf{i}}) = \min_{\mathbf{i}\in\mathbf{I}} J(\mathbf{i}) \qquad J(\mathbf{i}) = D(\mathbf{i}) + \lambda R(\mathbf{i}) \tag{B.1}$$

Dabei ist λ eine nichtnegative reelle Zahl, $D(\mathbf{i})$ die durch die Bitzuweisung entstandene Distortion und $R(\mathbf{i})$ die Anzahl der benötigten Bits.

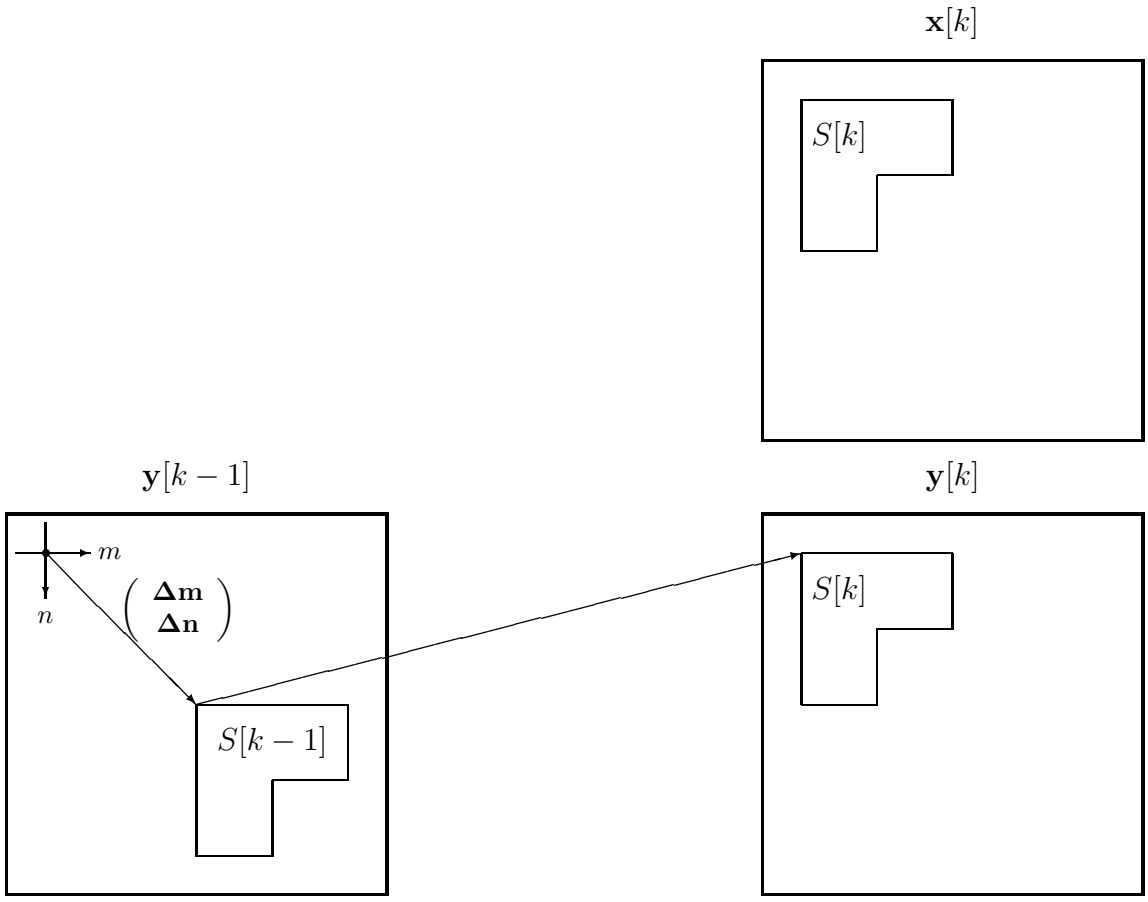


Abbildung B.12: *Veranschaulichung der Position des Segments in den einzelnen Bildern*

Für die Codierung eines Bildes ist die Intensitätsdichtedifferenz bei gegebener örtlich-zeitlicher Verschiebung $(\mathbf{\Delta m}, \mathbf{\Delta n})$ und Segmentform S optimal zu wählen. Dazu werden die Mittelwerte der Intensitätsdichten zwischen dem aktuellen Originalbild und der vorherigen Rekonstruktion bestimmt und quantisiert. **Abbildung B.12** veranschaulicht die Position der Segmente in den einzelnen Bildern.

$$\hat{\mathbf{q}}[k] = Q\left\{\frac{1}{|S[k]|}\sum_{(m,n)\in S[k]} \mathbf{x}[m,n,k] - \mathbf{y}[m+\mathbf{\Delta m}, n+\mathbf{\Delta n}, k-1]\right\} \tag{B.2}$$

Die minimalen Lagrange-Kosten bei gegebener örtlich-zeitlicher Verschiebung und Segmentform setzen sich aus dem quadtratischen Fehler der kompensierten Intensitätsdichten und den Ratentermen für die Segmentform $\mathbf{l}(S)$, die örtliche Verschiebung $\mathbf{l}(\mathbf{\Delta m}, \mathbf{\Delta n})$ und die Intensitätsdifferenz $\hat{\mathbf{q}}$ zusammen.

$$J(S, \mathbf{\Delta m}, \mathbf{\Delta n}, \hat{\mathbf{q}}) = \sum_{(m,n)\in S[k]} |\mathbf{x}[m,n,k] - \mathbf{y}[m+\mathbf{\Delta m}, n+\mathbf{\Delta n}, k-1] - \hat{\mathbf{q}}|^2 + \lambda\left[\mathbf{l}(S) + \mathbf{l}(\mathbf{\Delta m}, \mathbf{\Delta n}) + \mathbf{l}(\hat{\mathbf{q}})\right] \tag{B.3}$$

Eine effektive Suche im Parameterraum ist durch Berücksichtigung oberer Schranken der Zielfunktion möglich. Ist eine obere Schranke der Lagrange-Kosten bekannt, so reduziert sich die Anzahl der möglichen Parametersätze, unter denen die Zielfunktion minimiert wird. Jeder neue Parametersatz, dessen Lagrange-Kosten kleiner ist als der momentane Wert, schränkt somit den Suchraum ein.

$$J(\mathbf{i}_1) > J(\mathbf{i}_2) > J(\mathbf{i}_3) > \cdots > J(\hat{\mathbf{i}}) \tag{B.4}$$

Im weiteren setzt sich die Zielfunktion J aus verschiedenen Summenanteilen zusammen (Ratenterm, quadratische Fehler jeder Zeile eines Blocks). Aus diesen partiellen Summen J_μ^p läßt sich nun eine monoton zunehmende Folge der Lagrange-Kosten definieren, die gegen die Lagrange-Kosten des Parametersatzes $J(\mathbf{i})$ konvergiert.

$$J_m(\mathbf{i}) = \sum_{\mu=1}^{m} J_\mu^p(\mathbf{i}) \qquad\qquad J_m(\mathbf{i}) \longrightarrow J(\mathbf{i}) \tag{B.5}$$

Die Folge ist monoton zunehmend, da die partiellen Summen positiv sind.

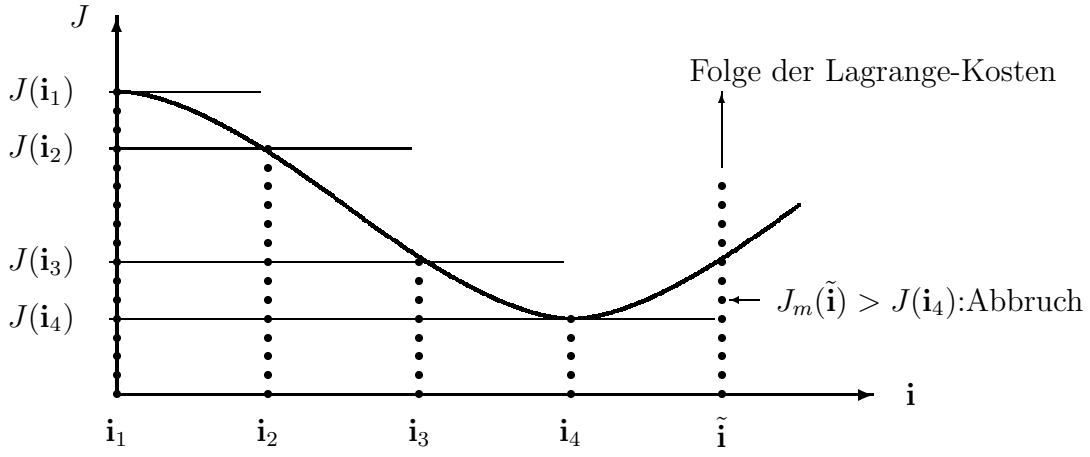


Abbildung B.13: *Beispiel für eine effektive Suche im Parameterraum mit oberer Schranke und abgebrochener Lagrange-Kosten-Folge*

Nun kann die Forderung, daß die Lagrange-Kosten des aktuellen Parametersatzes kleiner als die momentane obere Schranke zu sein haben, auf diejenige Forderung ausgedehnt werden, daß jedes Glied der Lagrange-Kosten-Folge kleiner als die monentane obere Schranke

zu sein hat. Überschreitet nun ein Glied der Folge die obere Schranke, so wird der aktuelle Parametersatz als möglicher Kandidat ausgeschlossen und die Folge abgebrochen. Dies ist gleichbedeuted mit einer Ersparnis an Rechenzeit, da partielle Summenanteile J_μ^p nicht berechnet werden müssen. **Abbildung B.13** veranschaulicht die effektive Suche im Parameterraum unter Berücksichtigung einer oberen Schranke der Zielfunktion.

B.2.6 `Growing`

Dieses Objekt bestimmt die allgemeine Quadtree-Struktur für alle Makroblöcke in den drei Farbkomponenten durch den *Growing-Algorithmus*.

```
int GrowQuadtree(QuadtreeNode *qtn, DesignData *dD){
  int l, k;

  /* design present node and all its children and */
  /* determine optimum shape, spatial- & intensity disp. for current node */
  if(!DesignNode(qtn, dD)) return(0);

  /* continue recursively */
  for(k=0; k<2; k++)
    for(l=0; l<2; l++)
      if(qtn->ndType[l][k])
        if(!GrowQuadtree(qtn->qtn[l][k], dD)) return(0);

  return(1);
}
```

Abbildung B.14: *Pseudo-Code für den rekursiven Growing-Algorithmus*

Die optimale Baumstruktur $\widehat{T^p}$ ist diejenige, die die geringsten Lagrange-Kosten unter allen möglichen verjüngten Teilbäumen des gesamten Baumes T aufweist. Ein verjüngter Teilbaum des gesamten Baumes $T^p \preceq T$ entsteht durch entfernen äußerer Zweige, wobei die Wurzel allen verjüngten Teilbäumen gemein ist.

$$J(\widehat{T^p}) = \min_{T^p \preceq T} J(T^p) \tag{B.6}$$

Um den Aufwand der Bestimmung aller verjüngten Teilbäume zu vermeiden, wird in diesem Objekt eine lokale Top-Down-Strategie angewandt. Diese Strategie wird rekursiv, beginnend mit der Wurzel, auf die Baumstruktur angewandt. **Abbildung B.14** zeigt einen Ausschnitt aus dem rekursiven Algorithmus. `DesignNode` entwirft für den aktuellen Knoten alle Kind-Knoten, die dabei als Endknoten behandelt werden, und bestimmt unter allen möglichen Knotenformen diejenige, die die Lagrange-Kosten minimiert. Die optimale Knotenform des aktuellen Knotens ermöglicht je nach Knotenform die rekursive Fortsetzung des Growing-Algorithmus bei den Kind-Knoten. Ist hingegen ein Endknoten die optimale Form des aktuellen Knotes, so wird die Rekursion nicht weitergeführt.

B.2.7 `Mask`

Das Objekt `Mask` wurde zur Verminderung der Blockeffekte, die bei der Kompensation auftreten, entworfen. Jeder kompensierte Pixel wird durch eine Maske mit neun Pixel

gewichtet. Erfahren die Pixel die gleiche Verschiebung, so werden die Effekte der Maske kompensiert. Dies hat Auswirkungen auf auf die Pixel eines Blocks. Da diese die gleiche örtlich-zeitliche Verschiebung erfahren, treten somit die Maskeneffekte nur an den Blockrändern auf.

Für eine effiziente Implementierung sind für die vorgesehenen Blockgrößen (16×16, 8×8, 4×4, 2×2, 1×1) die entsprechenden Masken bereits berechnet. Für die Blöcke an den Kanten und in den Ecken des Bildes werden diese explizit angegeben. Dabei wird die Position des Blocks relativ zum Rand des Bildes angegeben: innere Position, obere, untere, linke, linke obere, linke untere, rechte, rechte obere und rechte untere Position.

B.2.8 `Prediktor`

`Prediktor` beinhaltet Funktionen, die Schätzwerte für Bewegungsvektoren zu jedem Makroblock berechnen. Diese können dann je nach Option vom tatsächlichen Bewegungsvektor subtrahiert werden. Dies so entstandene Feld der Differenz-Bewegungsvektoren besitzt eine geringere Korrelation als das ursprüngliche Bewegungsvektor-Feld.

Bei der Berechnung der Schätzwerte wird dabei für jeden Makroblock nach den Farb-Komponenten differenziert:

- Der Schätzwert für die Y-Komponente berechnet sich aus dem Bewegungsvektor der obersten Ebene aus den direkten örtlichen Nachbarn.
- Der Schätzwert für die U- bzw. V-Komponente berechnet sich aus dem tatsächlichen Bewegungsvektor der obersten Ebene der Y-Komponente durch Integer-Division mit Zwei.

Da zuerst die Y-Komponente des Makroblocks bearbeitet wird und somit der tatsächliche Bewegungsvektor bekannt ist, ist diese Differenzierung möglich. Die Verwendung des Prädiktors ist eine Option, die sowohl am Coder als auch am Decoder gleich gewählt werden sollte.

B.2.9 `Pruning`

Das Objekt `Pruning` bestimmt für alle Makroblöcke in den drei Farbkomponenten die allgemeine Quadtree-Struktur durch einen rekursiven *Pruning-Algorithmus*.

Die optimale Baumstruktur $\widehat{T^p}$ ist diejenige, die die geringsten Lagrange-Kosten unter allen möglichen verjüngten Teilbäumen des gesamten Baumes T aufweist. Ein verjüngter Teilbaum des gesamten Baumes $T^p \preceq T$ entsteht durch entfernen äußerer Zweige, wobei die Wurzel allen verjüngten Teilbäumen gemein ist.

$$J(\widehat{T^p}) = \min_{T^p \preceq T} J(T^p) \tag{B.7}$$

In diesem Objekt wird eine lokale Bottom-Up-Strategie realisiert um den in diesem Sinne optimalen verjüngten Teilbaum zu bestimmen. Diese Strategie wird rekursiv, beginnend

```
int PruneQuadtree(QuadtreeNode *qtn, RecursionData *rD, Lagrangian *lgn){
  int l, k;

  /* continue recursively */
  for(k=0; k<2; k++)
    for(l=0; l<2; l++)
      if(qtn->ndType[l][k])
        if(!PruneQuadtree(qtn->qtn[l][k], rD, lgn)) return(0);

  /* pruned subtrees of present node determined */

  /* find optimum shape, spatial- & intensity displacement for current node */
  if(!QuantizeNode(qtn, rD, lgn)) return(0);

  return(1);
}
```

Abbildung B.15: *Pseudo-Code für den rekursiven Pruning-Algorithmus*

mit der Wurzel, auf die Baumstruktur angewandt. **Abbildung B.15** zeigt einen Ausschnitt aus dem rekursiven Algorithmus. Die Rekursion wird zunächst solange fortgesetzt, bis ein Endknoten des gesamten Baumes erreicht wird. Dieser wird nun mit `QuantizeNode` quantisiert, d.h. es werden die optimale Knotenform bzw. die minimalen Lagrang-Kosten bestimmt. Sind nun alle Zweige des darüber liegenden Mutter-Knotens determiniert, so kann der Mutter-Knoten selbst optimiert werden. Der Algorithmus wird solange fortgesetzt, bis wieder die Wurzel des Baumes erreicht ist. Damit ist der optimale verjüngte Teilbaum mit minimalen Lagrange-Kosten bekannt.

B.2.10 `Reconstruction`

Dieses Objekt ermöglicht die Kompensation der Modellparameter, d.h. der Bewegungsvektoren und Intensitätsdifferenz, für jedes Segment variabler Größe.

Bei der Vorwärts-Kompensation ohne Block-Überlappung werden die Blöcke aus dem vorherigen Bild unter Berücksichtigung der Transformationsvorschriften in das aktuellen Bild kopiert. Die implementierte Kompensation arbeitet rekursiv nach der Struktur des allgemeinen Quadtree wobei aber die Segmente in quadratische Blöcke zerlegt werden (**Abbildung B.16**).

$$\mathbf{y}[m,n,k] = \mathbf{y}[m+\mathbf{\Delta m}, n+\mathbf{\Delta n}, k-1] + \mathbf{q} \qquad (m,n) \in B[k] \tag{B.8}$$

Bei der Vorwärts-Kompensation mit Block-Überlappung werden die Blöcke aus dem vorherigen Bild unter Berücksichtigung der Transformationsvorschriften und unter Gewichtung der Pixelwerte mit einer nicht normierten Maske (**Abbildung B.17**) dem aktuellen Bild überlagert.

Nach der rekursiven Kompensation und Summation der Pixelwerte wird das gesamte Bild normiert, d.h. jeder Pixelwert wird durch 25 geteilt.

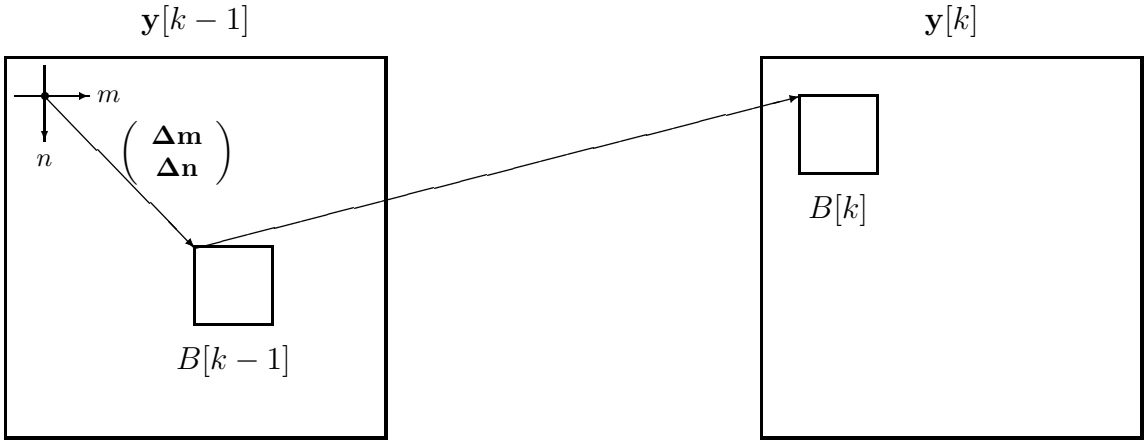


Abbildung B.16: *Vorwärts-Kompensation ohne Überlappung*

$$\begin{array}{ccc} 2 & 3 & 2 \\ 3 & \boxed{5} & 3 \\ 2 & 3 & 2 \end{array}$$

Abbildung B.17: *Nicht normierte Maske zur Gewichtung eines kompensierten Pixels*

B.2.11 `Skip`

Sehr häufig erfährt ein Makroblock keine Veränderung seiner Position oder Intensität. Um diesen Fall effektiv zu codieren, wurde dieser Fall abweichend vom Quadtree-Code implementiert.

Bevor eine Quadtree-Zerlegung eines Makroblocks für alle drei Farbkomponenten durchgeführt wird, berechnet dieses Objekt die Lagrange-Kosten des uncodierten Makroblocks für alle Farbkomponenten. Der Bewegungsvektor der Y-Komponente berechnet sich aus den Bewegungsvektoren der direkten örtlichen Nachbar-Makroblöcke. Für die U- und V-Komponenten werden die Werte der Y-Komponente durch zwei dividiert (Integer Division). Die Lagrange-Kosten setzen sich aus den quadratischen Fehler der Farbkomponenten und der notwendigen Datenrate von einem Bit pro Makroblock für alle Farbkomponenten zusammen.

$$\begin{aligned} J_{Skip}[k] \;=\; & \sum_{(m,n)\in B_Y[k]} \left|\mathbf{x}_Y[m,n,k] - \mathbf{y}_Y[m+\mathbf{\Delta m}_{YP}, n+\mathbf{\Delta n}_{YP}, k-1]\right|^2 + \\ & \sum_{(m,n)\in B_U[k]} \left|\mathbf{x}_U[m,n,k] - \mathbf{y}_U[m+\mathbf{\Delta m}_{UP}, n+\mathbf{\Delta n}_{UP}, k-1]\right|^2 + \\ & \sum_{(m,n)\in B_V[k]} \left|\mathbf{x}_V[m,n,k] - \mathbf{y}_V[m+\mathbf{\Delta m}_{VP}, n+\mathbf{\Delta n}_{VP}, k-1]\right|^2 + \lambda \cdot 1 \end{aligned} \tag{B.9}$$

Nach der Quadtree-Zerlegung des gleichen Makroblocks werden beide Lagrange-Kosten miteinander verglichen. Sind die Kosten des *Skip*-Falls kleiner als die der Quadtree-Zerlegung, so wird der Quadtree-Code verworfen und `COD=0` in die Codesequenz eingefügt.

B.2.12 `Structure`

Das Objekt `Structure` enthält eine effiziente Implementation der Makroblock-Struktur und beinhaltet Methoden um diese zu allokieren, zu löschen und zu initialisieren.

Der Makroblock enthält sowohl Daten für den *Skip*-Fall als auch die opimale Quadtree-Zerlegung aller Farbkomponenten. Jeder Knoten des Quadtrees enthält neben dem Formwert, Bewegungsvektor und Intensitätsdifferenz auch ihre korrespondierenden Codebuch-Indizes. Dadurch werden zeitintensive Konversionen vermieden. Blockzeiger auf das aktuelle Bild (`domain`) als auch auf das vorherige Bild (`range`) erlauben eine effiziente Blockbeschreibung ohne Bildinhalte zu kopieren. Die Strukturen sind im allgemeinen speicherintensiv implementiert um die daraus resultierenden Geschwindigkeitsvorteile zu nutzen.

B.2.13 `Zoom`

Dieses Objekt ermöglicht eine Interpolation eines Bildes durch Duplizierung der Pixelwerte. Dieses einfache Interpolationsschema genügt um die überlagerten Quadtree-Strukturen besser sichtbar zu machen denn die Segmentgrenzen sind ursprünglich nur ein Pixel breit. Als Interpolationsfaktoren sind nur natürliche Zahlen zugelassen.

Anhang C

Simulationsbedingungen

Die Beurteilung basiert auf den Testsequenzen „Car Phone“ und „Salesman“ mit jeweils einer Dauer von 10 s. Es werden die Testsequenzen sowohl in CIF-Auflösung als auch in QCIF-Auflösung bei 7.5 Vollbildern pro Sekunde untersucht.

Das erste Bild der Testsequenzen ist jeweils mit dem TMN 1.6 im *Intra-Mode* codiert. Dabei ist der Quantisierungsparameter auf 10 gesetzt. Die weiteren Bilder der Testsequenzen sind mit dem VBS-Codec bei konstantem λ bzw. mit dem TMN 1.6 bei konstantem Quantisierungsparameter im *Advanced Prediction Mode* codiert.

Zur Berechnung der Distortion addiert man den quadratischen Fehler zwischen Original $\mathbf{x}$ und Rekonstruktion $\mathbf{y}$ für alle Farbkomponenten und Bilder.

$$D_Y = \sum_{m=1}^{M} \sum_{n=1}^{N} \sum_{k=1}^{K} |\mathbf{x}_Y[m,n,k] - \mathbf{y}_Y[m,n,k]|^2 \tag{C.1}$$

$$D_U = \sum_{m=1}^{\frac{M}{2}} \sum_{n=1}^{\frac{N}{2}} \sum_{k=1}^{K} |\mathbf{x}_U[m,n,k] - \mathbf{y}_U[m,n,k]|^2 \tag{C.2}$$

$$D_V = \sum_{m=1}^{\frac{M}{2}} \sum_{n=1}^{\frac{N}{2}} \sum_{k=1}^{K} |\mathbf{x}_V[m,n,k] - \mathbf{y}_V[m,n,k]|^2 \tag{C.3}$$

Die Auflösung der Bilder in horizontaler bzw. vertikaler Richtung wird durch M bzw. N gekennzeichnet. Zu der Anzahl der codierten Bilder K zählt auch das erste Bild. Bei der Berechnung des $PSNR$-Wertes wird der quadratische Fehler schließlich auf die Anzahl der Bildpunkte $1.5 \cdot MNK$ normiert.

$$PSNR = 10 \log_{10} \left(\frac{255^2 \cdot 1.5 \cdot MNK}{D_Y + D_U + D_V} \right) \tag{C.4}$$

In der Literatur wird häufig für jedes Bild der $PSNR$-Wert berechnet und für die Sequenz der arithmetische Mittelwert der einzelnen $PSNR$-Werte ($\overline{PSNR}$) angegeben. Wird von logarithmischen Maßen zur Basis b ein arithmetischer Mittelwert gebildet, so werden die ursprünglichen Maße geometrisch gemittelt.

$$\frac{1}{K} \sum_{k=1}^{K} \log_b D[k] = \log_b \left(\prod_{k=1}^{K} D[k] \right)^{\frac{1}{K}} = \log_b D_g \tag{C.5}$$

Wir bevorzugen hingegen den arithmetischen Mittelwert nicht nur in Ortsrichtung sondern auch in Zeitrichtung.

$$D_a = \frac{1}{K} \sum_{k=1}^{K} D[k] \tag{C.6}$$

Der geometrische Mittelwert ist dabei immer kleiner oder gleich dem arithmetischen Mittelwert [27]. **Abbildungen C.1 und C.2** verdeutlichen den Unterschied zwischen arithmetischem und geometrischem Mittel in Zeitrichtung an den mit dem TMN 1.6 codierten Testsequenzen.

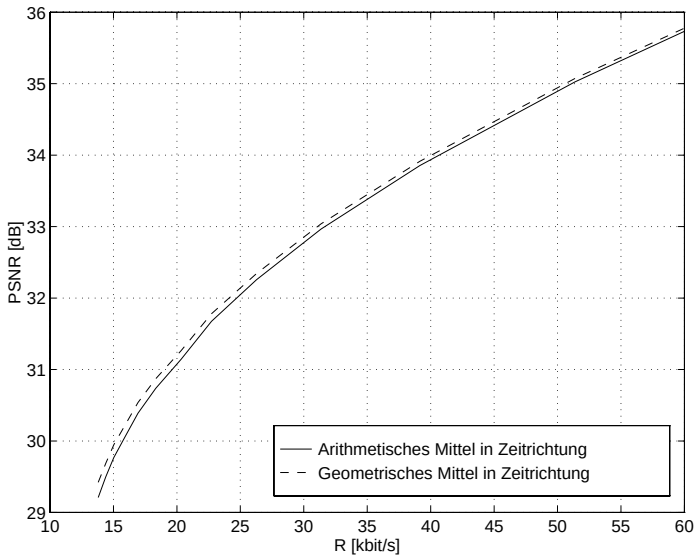


Abbildung C.1: *Vergleich zwischen arithmetischem und geometrischem Mittelwert in Zeitrichtung anhand der mit dem TMN 1.6 codierten Testsequenz „Car Phone“ (QCIF, 7.5 fps, 10 s).*

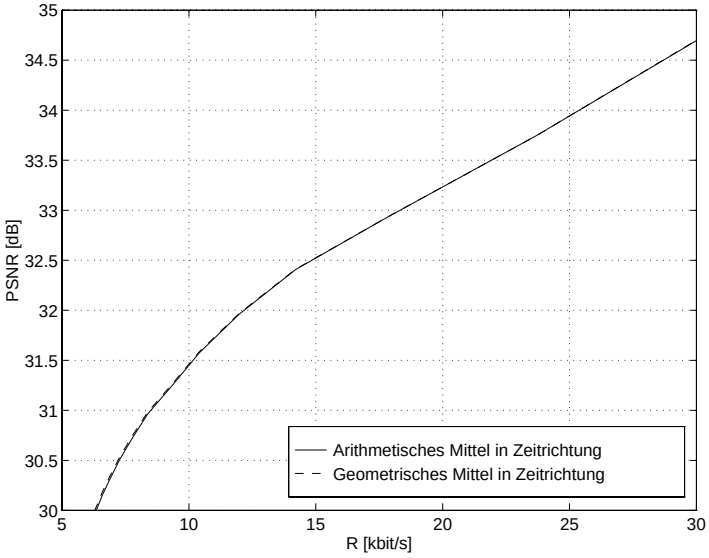


Abbildung C.2: *Vergleich zwischen arithmetischem und geometrischem Mittelwert in Zeitrichtung anhand der mit dem TMN 1.6 codierten Testsequenz „Salesman“ (QCIF, 7.5 fps, 10 s).*

Die Qualität der einzelnen Bilder der codieren Testsequenz „Car Phone“ scheinen stärker vom arithmetischen Mittel abzuweichen, da das geometrische vom arithmetischen Mittel differiert. Für die Testsequenz „Salesman“ ist das arithmetische und das geometrische Mittel annähernd gleich; die Qualität der einzelnen Bilder scheint der des Mittels zu gleichen.

Abbildung C.3 zeigt eine geometrische Interpretation für arithmetischen und geometrischen Mittelwert.

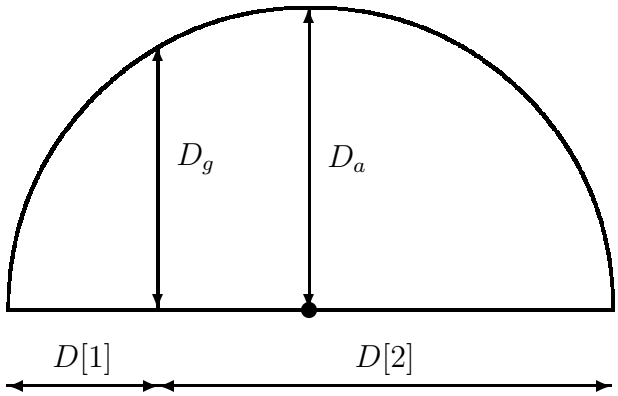


Abbildung C.3: *Veranschaulichung des arithmetischen Mittels* $D_a = \frac{1}{2}(D[1] + D[2])$ *und des geometrischen Mittels* $D_g = (D[1]D[2])^{\frac{1}{2}}$ *am Halbkreis mit dem Durchmesser* $D[1]+D[2]$*. Der geometrische Mittelwert ist immer kleiner oder gleich dem arithmetischen Mittelwert.*

Nehmen wir zum Beispiel eine Sequenz an, dessen erstes Bild dem Original entspricht ($D[1] = 0$) und dessen weitere Bilder stark verrauscht sind. Für den geometrischen Mittelwert erhält man $D_g = 0$ und somit für $\overline{PSNR} = \infty$. Das geometrische Mittel suggeriert in diesem Fall eine hohe Qualität vergleichbar mit der Originalsequenz obwohl dies tatsächlich nur für das erste Bild zutrifft. Der arithmetische Mittelwert D_a ist hingegen nur um den Faktor $\frac{K-1}{K}$ kleiner als der arithmetische Mittelwert D'_a der Sequenz ohne das ersten Bild. Der $PSNR$-Wert erhöht sich dabei nur um $10 \log_{10} \left(\frac{K}{K-1}\right) dB$ und ist für wachsendes K zu vernachlässigen.

Anhang D

Mathematische Modelle

D.1 Signaldarstellung und Systemkomponenten

$\mathcal{V}$ bezeichne die Menge aller Videosequenzen. Die Quelle sei beschrieben durch eine Zufallsvariable

$$\mathbf{X} : \mathcal{V} \rightarrow \{\mathbf{x}\} \tag{D.1}$$

bzw. die Rekonstruktion durch die Zufallsvariable

$$\mathbf{Y} : \mathcal{V} \rightarrow \{\mathbf{y}\} \tag{D.2}$$

Alle möglichen Videosequenzen der Quelle sind gleichwahrscheinlich:

$$f_{\mathbf{X}}(\mathbf{x}) = const. \qquad \forall \mathbf{x} \tag{D.3}$$

mit der Wahrscheinlichkeitsfunktion

$$f_{\mathbf{X}}(\mathbf{x}) = P(\mathbf{X} = \mathbf{x}). \tag{D.4}$$

Die Einheit Coder/Decoder kann man als Testkanal auffassen und beschreiben ihn mit der bedingten Wahrscheinlichkeitsfunktion $f_{\mathbf{Y}|\mathbf{X}}(\mathbf{y}|\mathbf{x})$.

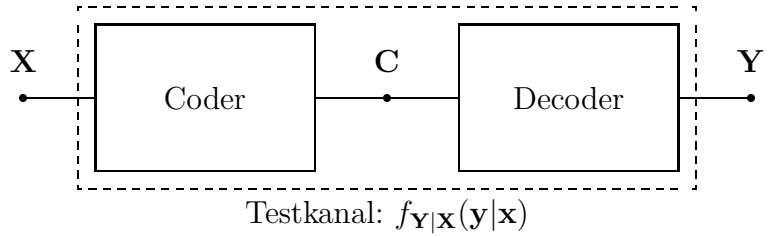


Abbildung D.1: *Modell des äquivalenten Testkanals*

$\mathcal{C}$ bezeichnet die Menge aller Codesequenzen. Der Code sei beschrieben durch die Zufallsvariable

$$\mathbf{C} : \mathcal{C} \rightarrow \{\mathbf{c}\} \tag{D.5}$$

Der Coder bildet die Videoquelle auf den Code, der Decoder den Code auf die Videosenke ab. Dabei besitzt der Decoder die Eigenschaft einer eineindeutigen Abbildung.

D.2 Diskretisierung einer kontinuierlichen Videosequenz

Die Videosequenz wird durch drei reellwertige Parameter (Ort x, Ort y, Zeit t) beschrieben:

$$\mathbf{x} := \mathbf{x}(x, y, t) \qquad (x, y, t) \in \mathcal{R}^3 \tag{D.6}$$

Wir führen eine Diskretisierung der kontinuierlichen Videosequenz $\mathbf{x}(x, y, t)$ durch. Die Diskretisierung beschreiben wir durch einen Basiswechel in ein ON-System. Für die Basis Ψ gilt folgende Orthonormalbedingung:

$$\begin{aligned}&\frac{1}{\Delta x \Delta y \Delta t} \int_{\mathcal{R}^3} \Psi(x - m\Delta x, y - n\Delta y, t - k\Delta t)\Psi(x - i\Delta x, y - j\Delta y, t - l\Delta t)d(x, y, t)\\ &= \delta[m, i]\delta[n, j]\delta[k, l].\end{aligned} \tag{D.7}$$

Die Koeffizienten der diskreten Videosequenz erhält man durch

$$\mathbf{c}[m, n, k] = \frac{1}{\Delta x \Delta y \Delta t} \int_{\mathcal{R}^3} \mathbf{x}(x, y, t)\Psi(x - m\Delta x, y - n\Delta y, t - k\Delta t)d(x, y, t). \tag{D.8}$$

Die Rekonstruktion der kontinuierlichen Videosequenz aus der diskreten ergibt sich zu

$$\mathbf{y}(x, y, t) = \sum_{(m,n,k)\in\mathcal{Z}^3} \mathbf{c}[m, n, k]\Psi(x - m\Delta x, y - n\Delta y, t - k\Delta t). \tag{D.9}$$

Die Diskretisierung einer kontinuierlichen Videosequenz kann man als „Codierung“ auffassen und soll die Signaldarstellung und die Systemkomponenten beispielhaft erläutern.

Literaturverzeichnis

[1] B. Girod, "The efficiency of motion-compensating prediction for hybrid coding of video sequences", *IEEE Journal on Selected Areas in Communications*, Band SAC-5, Num. 7, S. 1140–1154, Aug. 1987.

[2] B. Girod, "Motion compensation: Visual aspects, accuracy, and fundamental limits", in *Motion Analysis And Image Sequence Processing*, M.I. Sezan und R.L. Lagendijk, Ed. Kluwer Academic Publishers, Boston, 1993.

[3] B. Girod, "Motion-compensating prediction with fractional-pel accuracy", *IEEE Transactions on Communications*, Band 41, Num. 4, S. 604–612, Apr. 1993.

[4] Inc. Iterated Systems, "Mpeg-4 video submission, technical description", Document MPEG 96/0553, ISO/IEC JTC1/SC29/WG11, Mărz 1996.

[5] T. Wiegand und M. Flierl, "Entropy Constrained Variable Block Size Coding", Document MPEG 96/1012, ISO/IEC JTC1/SC29/WG11, Juli 1996.

[6] T. Wiegand und M. Flierl, "Entropy Constrained Variable Block Size Coding", Document MPEG 96/1291, ISO/IEC JTC1/SC29/WG11, Sept. 1996.

[7] P. Strobach, "Tree-structured scene adaptive coder", *IEEE Transactions on Communications*, Band 38, Num. 4, S. 477–486, Apr. 1990.

[8] J. Lee, "Optimal quadtree for variable block size motion estimation", in *Proceedings of the IEEE International Conference on Image Processing*, Okt. 1995, S. 480–483.

[9] M. Lightstone und S.K. Mitra, "Quadtree optimization for image and video coding", Chromatic Research, Inc, Feb. 1996.

[10] P.A. Chou, T. Lookabough und R.M. Gray, "Optimal pruning with application to tree-structured source coding and modeling", *IEEE Transactions on Information Theory*, S. 299–315, Mărz 1989.

[11] E. Riskin, "A greedy tree growing algorithm for the design of variable rate vector quantizers", *IEEE Transactions on Signal Processing*, Band 39, Num. 11, S. 2500–2506, 1991.

[12] T. Berger, *Rate Distortion Theory: A Mathematical Basis for Data Compression*, Prentice-Hall, Englewood Cliffs, NJ, 1971.

[13] R. Johannesson, *Informationstheorie - Grundlage der (Tele-) Kommunikation*, Addison-Wesley, Lund, 1992.

[14] H. Everett III, "Generalized lagrange multiplier method for solving problems of optimum allocation of resources", *Operations Research*, Band 11, S. 399–417, 1963.

[15] P.A. Chou, T. Lookabaugh und R.M. Gray, "Entropy-constrained vector quantization", *IEEE Transactions on Acoustics, Speech and Signal Processing*, Band 37, S. 31–42, Jan. 1989.

[16] Y. Linde, A. Buzo und R.M. Gray, "An algorithm for vector quantizer design", *IEEE Transactions on Communications*, Band COM-28, S. 84–95, Jan. 1980.

[17] ITU-T, *Draft Recommendation H.263 (Video Coding for Low Bitrate Communication)*, Okt. 1995.

[18] A.M. Tekalp, *Digital Video Processing*, Prentice Hall, London, 1995.

[19] M.T. Orchard und G.J. Sullivan, "Overlapped block motion compensation: An estimation-theoretic approach", *IEEE Transactions on Image Processing*, Band 3, Num. 5, S. 693–699, Sept. 1994.

[20] M. Lightstone, K. Rose und S.K. Mitra, "Locally optimal codebook design for quadtree-based vector quantization", in *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, 1995, S. 2497–2482.

[21] G.J. Sullivan und R.L. Baker, "Efficient quadtree coding of images and video", *IEEE Transactions on Image Processing*, Band 3, S. 327–331, Mai 1994.

[22] W.J. Zeng, Y.F. Huang und S.C. Huang, "Two greedy tree growing algorithms for designing variable rate vector quantizers", *IEEE Transactions on Circuits and Systems for Video Technology*, Band 5, Num. 3, S. 236–242, Juni 1995.

[23] P.A. Chou, M. Effros und R.M. Gray, "A vector quantization approach to universal noiseless coding and quantization", *IEEE Transactions on Information Theory*, Band 42, Num. 4, S. 1109–1138, Juli 1996.

[24] M.J. Sabin und R.M. Gray, "Global convergence and empirical consistency of the generalized lloyd algorithm", *IEEE Transactions on Information Theory*, Band 32, Num. 2, S. 148–155, Mărz 1986.

[25] A.-T. Schreiner, *Objektorientierte Programmierung mit ANSI C*, Carl Hanser Verlag, München, 1994.

[26] M.F. Barnsley und L.P. Hurd, *Fractal Image Compression*, A.K. Peters, Wellesley, Massachusetts, 1993.

[27] I.N. Bronstein und K.A. Semendjajew, *Taschenbuch der Mathematik*, Verlag Harri Deutsch, Frankfurt/Main, 1991.